2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

Synchronized Progress in Interconnection Networks (SPIN) : A New Theory for Deadlock Freedom

Aniruddh Ramrakhyani
School of ECE, Georgia Tech
aniruddh@gatech.edu

Paul V. Gratz
Dept. of ECE, Texas A&M University
pgratz@tamu.edu

Tushar Krishna
School of ECE, Georgia Tech
tushar@ece.gatech.edu

***Abstract*—One of the most fundamental design challenges in any interconnection network is that of routing deadlocks. A deadlock is a cyclic dependence between buffers that renders forward progress impossible. Deadlocks are a necessary evil and almost every on-chip/HPC network today avoids it either via routing restrictions across physical channels (Dally's Theory) or with at least one *escape* virtual channel (Duato's Theory). This ensures that a cyclic dependence between buffers is never created in the first place. Moreover, each solution is tied to a specific topology, requiring an updated policy if the topology were to change. Alternately, solutions have also been proposed to reserve certain resources (buffers) and allocate them only upon detection of a deadlock, thereby breaking the dependence chain and recovering from the deadlock. Unfortunately, all these approaches fundamentally lead to a loss in available bandwidth due to routing restrictions or buffer resource usage restrictions.**

In this work, we challenge the theoretical notion of viewing deadlocks as a lack of routing resource (buffers) problem that every solution to date is based on. We argue that a deadlock can in fact be considered as a lack of coordination between distributed entities. We prove that orchestrating a forward movement of *every* flit in the deadlocked ring at exactly the same time, which we call a *spin*, can guarantee forward progress and eventually lead to deadlock resolution with a bounded number of spins. We name this novel theory as SPIN (Synchronized Progress in Interconnection Networks). SPIN eliminates the need for virtual channels to achieve deadlock freedom thereby enabling fully adaptive routing with only one buffer per message class. We illustrate this capability by designing FAvORS, a novel truly one VC fully-adaptive routing algorithm. We also present a low-cost distributed implementation of SPIN and compare it against state-of-the-art deadlock avoidance/recovery schemes. SPIN provides up to 80% higher throughput, 52% lower area and 50% lower power for an on-chip 64-core mesh, and up to 83% higher throughput, 53% lower area and 55% lower power for an off-chip 1024-node dragon-fly.

***Keywords*—Networks on chip; Interconnection Networks; Deadlocks; Mesh; Dragon-fly**

I. Introduction

Increasing demand for compute power has led to the proliferation of Interconnection Networks in datacenters [1], [2], supercomputers [3] and compute chips [4]. Given a network topology, the key requirement for any network routing algorithm is that of freedom from deadlocks. A network deadlock is defined as a cyclic buffer dependency chain where each packet in the chain is waiting to acquire a buffer resource held by some other packet in the chain such that no forward progress is possible making the system unusable.

The problem of routing deadlocks in interconnection networks has received significant attention from the research community and plethora of schemes have been suggested for achieving deadlock freedom while minimizing their impact on performance at the same time [5], [6]. All prior research addressing the problem of routing deadlocks has considered deadlocks to be a resource dependency problem (buffers in this case). In other words, a flit sitting in a buffer is unable to move forward to the downstream router because there is no free buffer available, and a deadlock occurs when this dependence cycles back to the buffer of the original flit.

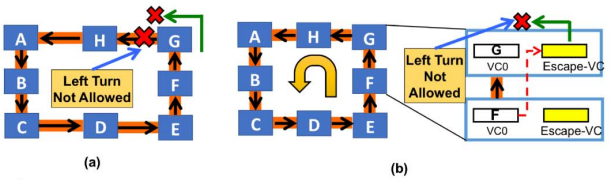


Fig. 1: **Deadlock Freedom with (a) Dally's (acyclic CDG) and (b) Duato's (escape VC) theories. Dally's theory places turn restrictions in all VCs while Duato's places them in only escape-VC.**

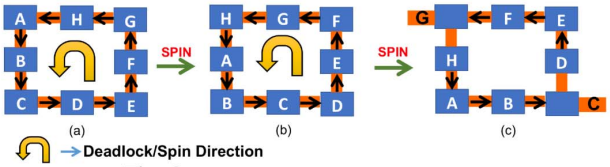


Fig. 2: **Deadlock Freedom with SPIN.**

At the outset, we would like to distinguish between the *theoretical framework* for deadlock freedom, and its *implementation*, acknowledging that there can be multiple implementations for the same theory. We classify prior implementations into four theoretical frameworks:

- *Dally's theory* [7] defines a strict order in acquisition of links and/or buffer resources/Virtual Channels (VCs) by network packets which ensures that a cyclic dependence is never created, as shown in Fig. 1(a). Here Packet G (sitting at south port of router) is not allowed to make a left turn thereby preventing a cyclic buffer dependence.
- *Duato's theory* [8] introduces the idea of escape paths (implemented as an additional set of buffers) that packets in a cyclic dependence can use to avoid or recover from deadlocks. These are known as *escape VCs* in network parlance, and are shown in Fig. 1(b). Routing inside regular VCs has no turn restrictions and can lead to a cyclic dependence; but routing inside the escape-VCs has

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00064

turn restrictions (for e.g., left turn is disallowed in Fig. 1(b)). Thus, packet F can move to the escape-VC of the downstream router and break the cyclic dependence.

- *Flow control* based schemes [9] restrict packet injection when the number of empty buffers in routers falls below a threshold to ensure there is at least one free buffer in any dependence chain, thus guaranteeing forward progress.
- *Deflection* routing [10] forces all flits to move every cycle even if they get misrouted in the process. The routing's inherent deadlock free nature is due to forced packet movement every cycle (though livelocks must be independently addressed).

Any deadlock-free system today (in products or in research) uses one of these four as the underlying theory guaranteeing deadlock-freedom. Dally's theory and deflection directly avoid deadlocks. Duato's theory and flow-control can either be used to avoid deadlocks or recover from them.

In this work, we introduce an *alternate theoretical framework* for deadlock freedom called SPIN (Synchronized Progress in Interconnection Networks), with the following working principle: if every router in a deadlocked cycle could send out its blocked flits at *exactly the same time*, without first requiring the downstream buffer to become free, there will be safe forward progress. We call this a *spin*. Within a provable upper bound in the number of spins, one of the packets would exit the deadlocked loop thereby providing deadlock freedom. We show an example in Fig. 2. In Fig. 2(a) packets $A - H$ are in a deadlock, as highlighted by the edges between routers which represent the direction that each packet wants to travel to for its next hop. After two spins, (Fig. 2(b) and (c)), packets *G* and *C* exit the cycle thereby breaking the deadlock.

This paper makes the following key contributions:

- We present SPIN, a novel theoretical framework for deadlock freedom, with proof of correctness. SPIN is fundamentally different from all prior theories in that it views deadlocks as a lack of communication/coordination, rather than as lack of buffers problem, making it a valuable addition to interconnection network literature.
- We compare SPIN qualitatively against the four existing frameworks in literature and provide an insight into when it makes sense to use SPIN over others.
- We present a low-cost, topology-agnostic, distributed implementation of SPIN. We quantitatively compare it against implementations of alternate frameworks.
- We design a first of its kind *one-VC fully adaptive deadlock free routing algorithm* called FAvORS that leverages SPIN for providing deadlock freedom. We profile its performance on a mesh and dragonfly topology with just 1 VC without adding any routing or flow control restrictions which is impossible across known solutions today.

The key benefit of SPIN is that it is topology agnostic, making it highly flexible compared to all prior frameworks. Moreover, our proposed implementation performs the recovery process in a fully distributed manner, allowing it to scale well to large networks. Thus, we would like to emphasize that we see SPIN as a promising choice for guaranteeing deadlock freedom for irregular/application specific interconnection networks such as random graph based datacenter topologies like Jellyfish [11], static and dynamically changing irregular on-chip topologies occurring as a consequence of faulty/power-gated network elements [12], [6], NoC generators [13], and on-chip fabrics within domain-specific accelerators [14], [15]. All these cross-domain interconnection networks can plug in SPIN to provide deadlock free adaptive routing without requiring additional VCs/buffers to avoid routing deadlocks[1].

The rest of the paper is organized as follows: Sec. II provides relevant background and related work, Sec. III presents the theory behind SPIN, while Sec. IV discusses an implementation with a walk-through example and demonstrations of correctness/validation. Sec. V describes our 1-VC fully adaptive routing algorithm. Sec. VI presents the evaluations and Sec. VII concludes.

II. Background and Related Work

In this section, we describe the deadlock freedom theories and recent works based on these theories.

A. Dally's Theory

Dally et. al. [7] introduced the concept of using a *Channel Dependency Graph* (CDG) to model buffer resource dependencies in the network and define a sufficient condition for deadlock freedom: an acyclic CDG. In essence, an acyclic CDG establishes a total order in the acquisition of buffer resources by network packets. A network with cyclic CDG can be made deadlock free by splitting each physical channel that is part of the cycle/s into a group of VCs and defining a strict priority order in the acquisition of these VCs such that the extended CDG becomes acyclic.

A number of routing algorithms [17] have been proposed that use Dally's theory. These algorithms rely on either routing restrictions at the cost of reduced routing adaptivity and fault tolerance [5], [20], [21] or provide additional VCs [16], [22], [17] for full/partial routing adaptivity at the cost of increased router area and energy consumption. The turn model [5] is the most popular direct application of Dally's theory. It prohibits a subset of the legal turns a packet can make in the network just enough to make the CDG acyclic (e.g. the deterministic *XY* routing or partially adaptive *West-first* routing). Fully adaptive routing algorithms that use Dally's theory for deadlock freedom provision additional VCs to provide routing adaptivity [17].

Deadlock avoidance schemes based on Dally's theory suffer from increased packet latency and throughput loss due to routing restrictions, and/or increased energy consumption and area requirements due to high number of VCs [6]. In addition, the complexity of finding and removing cycles in the CDG whose size scales exponentially with the number of VCs limits the theory's use to small networks [8].

[1]The buffer overhead in SPIN is a single buffer in the control path to store the deadlock path as we discuss later. The framework does not add any additional buffer/VC in the router datapath.

TABLE I: Comparison of Deadlock Freedom Theories

Theory	Packet Injection / Scheduling Restrictions	Acyclic CDG Required	Topology Dependent	VC cost for :				Livelock Freedom cost
				Minimal Deterministic Routing		Fully Adaptive Routing		
				Mesh (NxM)	**Dragon-fly**	**Mesh** (NxM)	**Dragon-fly**	
Dally's Theory	No	Yes	Yes	1	2 [16]	6 [17]	3 [16]	None
Duato's Theory	No	No*	Yes**	1	2	2	3	None
Flow Control	Yes	No	Yes	2 [18]	2 [19]	2 [18]	2 [19]	None
Deflection Routing	Yes†	No	No	Not possible‡	Not possible‡	0◇	0◇	High
SPIN	No	No	No	1	1	1	1	None

* Only an acyclic connected sub-graph
** Need to know topology to design acyclic CDG within the escape virtual channel.
† Cannot inject if number of packets currently at router is equal to the number of its output ports
‡ Minimal routing cannot be guaranteed by design
◇ Cost is 0 assuming bufferless design. Routing is adaptive in the sense that it prevents network hotspots from forming due to deflections. However, deflections may be out of unfavorable ports so routing is not "fully adaptive".

B. Duato's Theory

Duato's theory [8] guarantees deadlock freedom for networks with or without cycles in their CDG as long as there exists a connected sub-graph in the extended CDG that is acyclic. It splits each physical channel into a set of additional VCs (i.e., buffers) to form an *escape network*. Packets in the regular VCs are routed adaptively while those in escape VCs get routed using a turn restriction based deadlock free routing algorithm. Duato's theory can be used for both deadlock-avoidance [23], [24], [8] and deadlock-recovery [25], [26], [27], [28], [29].

Duato's theory enables deadlock free fully adaptive routing at significantly reduced VC cost (only two for a 2-D Mesh) compared to Dally's theory, However, it suffers from (a) energy and area overheads of escape network buffers, and (b) additional routing tables/logic to support deadlock free routing within the escape VCs.

C. Flow Control

Flow control techniques restrict the rate/time of packet injection and/or delay packet routing to prevent network deadlocks. Bubble Flow Control (BFC) [9], a popular flow control technique, guarantees deadlock freedom for a ring topology if there exists at least one free buffer/VC in the ring at all times. This technique has been extensively used to provide deadlock freedom for Torus networks [30], [31]. VCs are divided into two sets : regular and escape, similar to the division used in *Duato's theory*. Fully adaptive routing is used in regular VCs while packets are routed using Dimension Ordered Routing (DOR) with BFC within the escape VC.

Canwen et al. [18] and Garcia et al. [19] leverage BFC within a 2-D mesh and a dragon-fly. *Static Bubble* [6] leverages BFC within dynamically changing irregular topology derived from the mesh topology. Critical Bubble Scheme (CBS) [32] implements global BFC by marking one packet sized VC in each ring as Critical Bubble. The critical bubble flows backwards as the packets move forward in the ring. Worm-Bubble Flow Control (Worm-BFC) [31] extends CBS to wormhole routing. Buffers are colored as grey, white and black and serve as token for routers ensuring the presence of critical bubble in the ring at all times thereby preventing starvation and deadlock.

Flow-control based schemes suffer from limitations like buffer coloring/token capturing complexity, expensive solutions required for preventing packet injection starvation particularly for wormhole routing, energy overhead of circulating the token/buffer color information and lack of guarantees for packet latency that have prevented their adoption in commercial and academic designs.

D. Deflection Routing

Deflection routing [10] or *Hot-potato* [33] routing eliminates routing buffers by requiring routers to assign every input flit to some output port every cycle. When more than one flit requests the same output port, only one (chosen according to a priority scheme) is allotted the output port and the rest are deflected to some other available output port [10]. Each flit however is assigned a unique output port and is not buffered. This idea, and its variants has been leveraged by *BLESS* [10], *MinBD* [34], *CHIPPER* [35], Jafri et al. [36] and others.

Deflection routing is a not a deadlock-freedom theory in itself; its inherent deadlock-free nature is a result of the observation that for any given router with n output ports, there will always be some matching of up to n input packets to the n output ports such that packet movement can be ensured without causing deadlock (though not always forward progress). Deflection routing however, suffers from major limitations, including requiring livelock freedom solutions, large reassembly buffers for out of order packet delivery and lack of guarantees on packet latency. In addition, it offers lower saturation throughput compared to buffered routing algorithms and higher packet latency and network energy consumption at high loads due to misrouting [36].

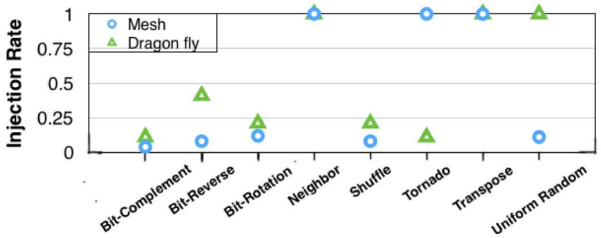


Fig. 3: **Minimum injection rate (flits/node/cycle) at which 64-core Mesh and 1024-node Dragon-fly deadlock with different traffic patterns in 100K cycles with 3 VCs per port and 1-flit packets.**

E. Comparison with SPIN

In Table I, we provide a qualitative comparison of our proposed framework with other deadlock freedom solutions. We highlight a few key attributes of SPIN. First, SPIN is completely topology agnostic (i.e., does not require any knowledge about the CDG), making it one of the most flexible frameworks to date. We would like to emphasize that Dally's theory and Duato's theory can work on arbitrary topologies; however, their implementations inherently require a knowledge of the CDG (i.e., network topology) at design-time [25], [5], or reconfiguration-time [20], [29] in order to construct a deadlock-free path via the physical channels (Dally's theory) or escape virtual channels (Duato's theory). Second, SPIN enables *fully adaptive routing* with *one VC* - the least VC-cost theoretically possible for any buffered routing algorithm[2]. This is a significant capability as VCs have high energy and area overheads and using them solely to avoid/recover out of rare events like routing deadlocks is expensive. Third, SPIN does not place any injection or packet routing restrictions, and is inherently livelock-free, unlike flow-control and deflection-based schemes.

Dally's theory and deflection routing avoid deadlocks from forming, while Duato's theory and flow-control routing have been used for both actively avoiding deadlocks [29], [9] and recovering out of them [25], [6]. We implement SPIN as a recovery mechanism[3], motivated by the premise that deadlocks are extremely rare events (as we demonstrate next) and thus placing resource reservations or resource-use restrictions (as prior deadlock avoidance schemes do [5], [20]) for rare events is counter-intuitive. We discuss overheads of our implementation of SPIN later in Sec. IV-D.

F. Routing Deadlock Occurrence in Mesh and Dragon-Fly

To illustrate that deadlocks are rare events, in Fig. 3 we plot the minimum injection rates at which an on-chip 64-core mesh and an off-chip 1024-node dragon-fly topologies with minimal routing and UGAL routing respectively deadlock at least once across different synthetic traffic patterns. The network interface controllers (NIC) eject flits without any stalls. All simulations were run for 100K cycles with 1-flit packets and 3-VCs per input port. We observe that the minimum injection rate across different traffic patterns at which these topologies start to deadlock is at-least 10x the injection rate of real applications [6] where network requests get filtered by L1 and L2 caches. In addition, for some traffic patterns like tornado and transpose, no deadlock is observed with minimal routing on a Mesh topology even at 100% injection rate due to the nature of these patterns.

[2]We note, however, that for all deadlock-freedom schemes in Table I including SPIN, more VCs may still be required to avoid protocol deadlocks.

[3]SPIN could be implemented as an avoidance scheme via proactive spinning, though we do not explore that in this work.

III. Theory behind SPIN

In this section, we discuss how SPIN's coordinated movement based recovery guarantees deadlock freedom.

Definition 1: Deadlock. A cyclic dependence chain between buffers, as depicted in Fig. 2(a).

Definition 2: Spin. A one hop movement of the deadlocked ring via a synchronized movement.

Definition 3: Forward Progress. A hop that brings a packet closer to its destination.

Theorem: *In a deadlocked ring of length m, at most k spins are required to resolve the deadlock, where k=$m-1$ in case of minimal routing, and k= $m \times p$ + (m-1) in case of non-minimal routing, where p is the maximum number of times a packet can be misrouted in the non-minimal routing algorithm.*

Proof: *Case I: Minimal Routing.* With minimal routing, every hop by definition leads to forward progress. Suppose we start with a deadlocked ring of length m. After m spins the packets would be back at their starting positions which would contradict the definition of minimal routing. By the $(m-1)$th spin, at least one of the packets will request for an output port that is not part of the dependence chain, as shown in Fig. 2(c), thereby breaking the deadlock.

Case II: Non Minimal Routing. In non-minimal routing, it is possible for packets to continue requesting the same output ports that lead to the deadlock, if more productive output ports are congested. This is the problem of livelocks. To avoid livelocks, non-minimal routing algorithms place a limit p on the number of times a packet can be misrouted. For example, in UGAL [37], p is 1. In the worst case, after $m \times p$ spins, no more misroutes will be allowed, and the scenario will be identical to Case I and will require at most $m-1$ more spins to resolve the deadlock.

Achieving Coordinated Movement: Any system implementing SPIN needs the following three features:

(I) Deadlock Detection

(II) Coordinating a single time to perform the SPIN

(III) Performing the actual SPIN (i.e., simultaneous one-hop forward movement by all packets in the deadlocked ring).

In small networks, it is not hard to design centralized implementations of these three features. For scalability, however, we present a fully-distributed implementation next.

IV. An Implementation of SPIN

We present a distributed topology-agnostic implementation of SPIN in this section, which works as is for both on-chip and

off-chip. It is possible to add topology-specific or on/off-chip specific optimizations. We note that there are many possible implementations of the SPIN theory, this is one.

Our implementation detects a deadlock using counters and confirms its presence by sending out a *probe* that traces the deadlock path. Once confirmed, a *move* msg is sent out to convey the *spin cycle* to deadlocked routers. At the *spin cycle*, all deadlocked routers transmit the deadlocked packets.

A. FSM for implementing the SPIN features

We add one counter with a *finite state machine* (FSM) at design time to every router in the network topology. The counter FSM has seven states as shown in the Fig. 4(a) where the state transitions are based on the phase of the deadlock recovery process. FSM transitions for the recovery initiating router are shown in the upper-half of Fig. 4(a) while the lower half shows the FSM transitions for other routers in the deadlocked chain. The counter operates on two thresholds (in cycles): t_{DD} ($_{DD}$ = deadlock detection) which is configurable, and t_{LL} ($_{LL}$ = loop length) which is the length of the deadlock loop this router is currently part of. To achieve coordinated movement of packets for deadlock recovery, we use *four* special messages (SMs): *probe, move, probe_move* and *kill_move*. These SMs use the router's regular links in a bufferless routing fashion, at higher priority than normal packets. As we will show later, this has negligible impact on the latency and throughput of the network due to the SMs' relative infrequency.

B. Walkthrough Example

We discuss the implementation using a walkthrough example in Fig. 4(b)-(d). Each buffer dependence in the figure is marked with the packet(s) that want to use it for the next hop. As can be seen, there exists a deadlock due to the following cyclic buffer dependency chain :
(A,B) ⇒ (C) ⇒ (E,F) ⇒ (G,H) ⇒ (I,J) ⇒ (K) ⇒ (A,B)
We discuss a Virtual-Cut-Through (VCT) implementation for simplicity though a wormhole design is also possible with some additional complexity.

The FSM starts in the S_{OFF} state. When a new flit arrives at a router (at any port other than the local ones), the counter state is changed to S_{DD} ($_{DD}$ here stands for "Deadlock Detection") with threshold set to t_{DD} (configurable) and counter starts pointing to the VC it occupies. If the flit leaves within the threshold time, the counter pointer is incremented to point to a non-empty VC in a round-robin manner and the counter is reset and restarted. If all VCs at the router are idle, the counter goes back to S_{OFF}.

1) Phase I: Deadlock Detection: The counters at all the nodes in the deadlock loop are in S_{DD} (Fig. 4(b)). The counter at node 5 expires in S_{DD} as packet *I* does not leave within the threshold time (**Step 1**). Node 5 then sends out a *probe* message (**Step 2**) from the North output port (output port for packet *I*) to confirm the presence of deadlock and rule out false positives due to congestion. The counter is reset and restarted with the same threshold t_{DD} and state S_{DD}.

At each router, if *all* VCs at the input port of the probe are active, the probe is forked out of all unique output ports that packets in the VCs at the input port are waiting on (except ejection). If all VCs are not active, the probe message is dropped. The fork operation creates identical copies, so all the information already present in the probe is preserved. Each forked copy is appended with the outport direction it leaves the router from. In this case, packet K wants to go West while packet Z wants to go East, hence the probe is forked out of these output ports (**Step 3**). Node 2 appends "W" to the probe that goes left and "E" to the probe that goes right.

At node 3, the probe is dropped (**Step 4a**) as both packets *M* and *N* are waiting to get ejected. Clearly, packets waiting for ejection cannot be a part of a cyclic buffer dependency chain inside the network.

At nodes 1, 4, 6 and 7, the probe is forwarded out of the south, south, east and north output ports (**Step 4b**), and the port-ids S, S, E and N respectively are appended to it.

When node 5 (the recovery initiating router), receives the probe back (**Step 5**), *the dependence chain is confirmed* and the path acquired by the probe is latched in a special buffer called the *Loop Buffer* (**Step 6**).

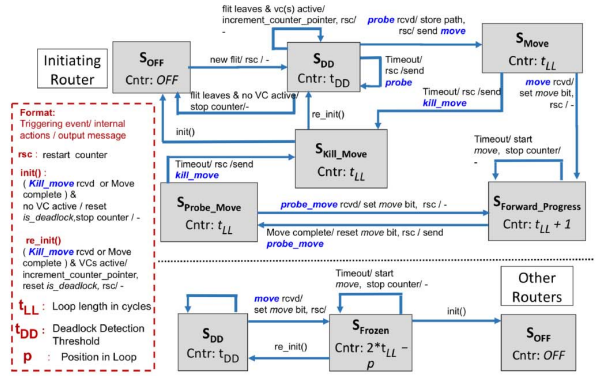
What if the counter at node 5, expires before the probe returns or all the copies of the probe are dropped? The counter at node 5 resets and restarts with the same threshold (t_{DD}) and state (S_{DD}), and the FSM sends out a new probe. This however cannot continue infinitely. If there is a deadlock, the probe would return. Else things may be moving slow due to congestion. Eventually, the congestion would clear up and the flit would leave and the counter would point to a new VC.

What if another node in the dependency chain, sends out a probe before/after it has received the probe from node 5? The probe SM only acquires the dependency path and does not alter the architectural state of the router. The state is altered upon receiving other SMs (described next). Thus, there is no functional correctness problem.

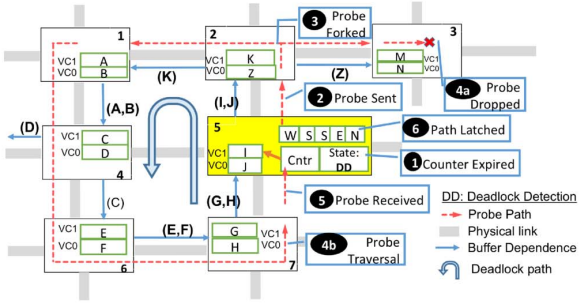
What if node 5 receives a probe from another node after it has sent out a probe? The fact that a router sent out a probe is neither reflected in the FSM state (FSM returns to same state (S_{DD}) after sending out the probe upon counter expiry) nor is buffered anywhere in the router. Node 5 would process this probe in the same way as its probe was processed by other nodes i.e. fork it out of all buffer dependencies at the input port and append corresponding outport ids.

2) Phase II: Communicating the SPIN time: The *SPIN* scheme uses coordinated movement of deadlocked packets for deadlock resolution. In essence, all routers in the cyclic dependency chain move the deadlocked packets out of the requested output ports in the same cycle together. This causes all the deadlocked packets to *spin* in the direction of the cyclic dependency chain simultaneously leading to one hop movement of all deadlocked packets. The recovery initiating router (node 5) sends out a SM (*move* SM) to convey the cycle time for spin to the deadlocked routers.

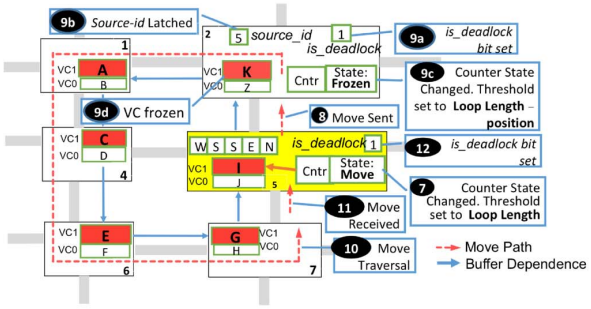
After receiving the *probe* back (Fig. 4(c)), the FSM at node 5 transitions to S_{Move} (**Step 7**) with the counter threshold set



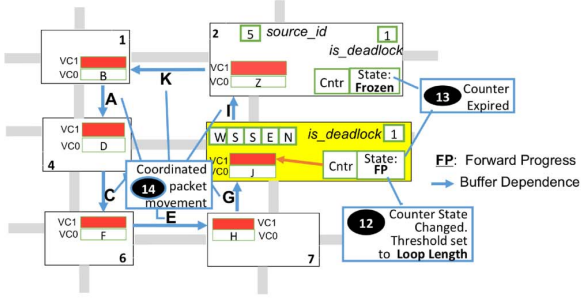
(a) FSM in every router for implementing SPIN.



(b) I: Deadlock Detection (*Probe* Traversal).



(c) II: Communicate SPIN cycle (*Move* Traversal).



(d) III: The SPIN

Fig. 4: **Walk-through Example and Router Microarchitecture of SPIN.**

to t_{LL} in cycles[4] and sends out a *move* SM (**Step 8**). The move message is guaranteed to return in this time unless it is dropped (explained later). The move message is embedded with the path of deadlock loop, the id of the sender and the cycle time for the *spin*.

Upon receiving the *move* SM, each router freezes one VC at the input port of the *move* SM that wants to use the output port of the *move* SM (and hence is a part of the deadlock dependency chain) by disabling switch allocation for it. The packet in this VC will leave the router only during the *spin* cycle. In addition, the counter FSM state is changed to S_{Frozen} with counter threshold set to count to the *spin* cycle. In the walkthrough example, upon receiving the *move* SM, node 2 sets its *is_deadlock* bit (**Step 9a**), latches the id of the sender in the *source-id* buffer (**Step 9b**) and changes the FSM state to S_{Frozen} (**Step 9c**). Node 2 extracts the first port-id (*West*) from the path embedded in the *move* SM and freezes packet *K* (which wants to go West) (**Step 9d**) for the spin. At each router, the first outport id is stripped away from the path embedded in the *move* SM, which ensures that the outport of the move SM corresponding to a node at which it is received is always the first, thereby speeding up the forwarding circuitry.

The *move* SM is processed in a similar manner at nodes 1, 4, 6 and 7 with the *move* SM being forwarded out of South, South, East and North outports respectively and packets A, C, E and G respectively being frozen for the *spin* (**Step 10**). Upon receiving the move message (**Step 11**) node 5 sets its *is_deadlock* bit and freezes packet *I*.

How is the *spin* cycle decided? The spin cycle is calculated as follows: (cycle in which *move* SM is sent out) + 2×(deadlock loop length in cycles)

Why is the spin cycle offset by 2×(deadlock loop length in cycles) ? It takes *loop length* number of cycles for the *move* SM to come back to its sender. The *spin* can be initiated as soon as all of the routers have processed the *move* SM and the confirmation has reached the sender. However, it may happen that the *move* SM is dropped at a router that is not able to honor the *spin* request (because the dependency detected earlier by the probe no longer exists at this router or it has accepted the move request from some other router). In this case, other routers that have accepted the *spin* request need to be informed to cancel the *spin* and unfreeze the packets. This is done through another SM : *kill_move* (see Sec. IV-B5) which would take another *loop length* number of cycles to traverse the loop.

Why is it necessary to freeze the packets? Rarely, due to the dynamically changing buffer dependency scenarios due to packet movement, some packet in the original detected deadlocked dependency chain may be able to make forward progress. This is true in cases when there are more than one VCs per input port and due to clearing up of some other congestion in the network some VC may free up enabling a

[4]The loop length (LL) is calculated from the path in the received probe.

deadlocked packet to make forward progress. In this case, the *spin* might push an invalid packet to some router or overwrite some other packet. To prevent this, a valid packet that is a part of the detected dependency chain is frozen.

3) Phase III: The SPIN: After receiving the *move* SM back (Fig. 4(d)), the FSM at node 5 transitions to $S_{\mathsf{Forward\_Progress}}$ and the counter is set to count to the *spin* cycle (**Step 12**). The counters at all the deadlocked routers expire together in the *spin* cycle (**Step 13**). Upon counter expiry, routers push out the flits of their *frozen* packets on the requested output links (**Step 14**) causing the deadlocked dependency chain to *spin*.

This completes the SPIN recovery process and the routers can now resume normal operation. However, a single spin may not be enough to break the deadlock. In the framework, the counter at some deadlocked router would expire again and the recovery process would be repeated. To speed up the case of deadlocks requiring multiple spins to get resolved, we introduce an optimization.

4) Optimization for multiple spins: After one spin is complete, the FSM at node 5 could potentially transition back to S_{DD} and wait for a timeout to send out a probe again in case the deadlock still exists and additional spins are required. As an optimization however, the FSM at node 5 transitions to $S_{\mathsf{Probe\_Move}}$ and sends out a *probe_move* SM along the path latched in the *loop buffer* at node 5 to check if the deadlock still exists. The *probe_move* performs the joint function of probe and move, and instructs routers to freeze a deadlocked packet (if the dependency still exists) till the *spin* cycle.[5] If the *probe_move* returns to the sender, it confirms the continued presence of deadlock and the *spin* is repeated.

In our walkthrough example, the *probe_move* would get dropped at node 4 as neither of the packets (A,D) want to use the south outport. Thus, the deadlock due to previously detected dependency chain has been resolved. Meanwhile, the counter at node 5 expires while waiting for the *probe_move* to return and transition to $S_{\mathsf{kill\_move}}$ (explained next).

5) Cancelling the SPIN (if required): If the *move* or *probe_move* SM gets dropped at a node, it indicates that the previously detected dependency chain no longer exists. In this case, routers that have processed the *move* or *probe_move* (before it was dropped) need to be informed to cancel the *SPIN*. This is done through another SM, the *kill_move* msg.

If the counter at the sender expires while waiting for *probe_move/move* to return, the FSM transitions to $S_{\mathsf{kill\_move}}$ and counter threshold is set to the length of deadlock path (in cycles). In our walkthrough example, node 5 would send out a *kill_move* SM (embedded with the loop path and sender id) along the path latched in the loop buffer to instruct routers to unfreeze the frozen VCs and resume normal operation.

After receiving the kill_move back, node 5 would clear the path buffer, reset the *is_deadlock* bit and resume normal operation. The FSM at node 5 will transition to S_{DD} if there are non-empty VCs at the router (which are not waiting to get ejected) else to S_{OFF}.

Our implementation provides distributed solutions for the three requirements of SPIN theory. In addition, it introduces additional optimization to speed-up deadlock recovery and handles false-positives. Next, we discuss design choices related to the correctness of the implementation.

C. Correctness

The correctness of our SPIN implementation is guaranteed because of two key constructs: (i) strict priorities among SMs and routers, and (ii) the FSM. We describe these next.

1) Correctness due to Priority Orders: **Priority among SMs:** The implementation discussed in the previous section provides a distributed design where any router in the deadlock chain can initiate deadlock recovery by sending out SMs. As a result, a router may receive more than one SM in the same cycle from its different ports if it is part of multiple dependency chains. It is thus important to define a strict priority order in processing of SMs to prevent races and ensure that all routers in a deadlocked chain maintain a consistent architectural state. Among the different SM classes, the priority is defined as: *probe_move* $>$ *move* **or** *kill_move* $>$ *probe* $>$ *flit* .

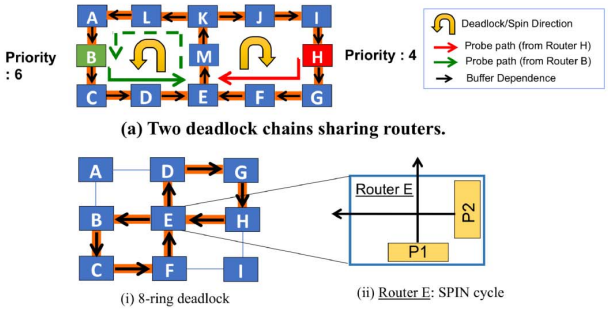
Also, if the router is a part of multiple dependency chains, it may receive *probes* from all such chains in the same cycle. In this case, we need to decide which *probes* to process (in case multiple want to use the same output port). This rule is given by the *"principle of rotating priority"*.

Principle of rotating priority among routers: For a network with N routers, the system starts with router N having the highest priority, down to router 1 having the lowest priority. After an *epoch*, the priorities are changed in round-robin manner. The length of *epoch* is critical. Too large value would delay deadlock resolution as probes from routers in the deadlock loop would get dropped by probes from higher priority nodes (not part of the deadlock loop) which may enter the loop due probe forking and keep looping due to the buffer dependency while too small of a value wouldn't guarantee deadlock resolution for arbitrary length deadlock loops.

For the SPIN scheme, we choose ($4 \times t_{\mathsf{DD}}$) as the length of the epoch. The round-robin priority update guarantees that every router in the network has the highest priority eventually and the epoch guarantees that the router will have highest priority long enough to detect a deadlock, send out a probe and receive it back, ensuring functional correctness. It is important to note here that the rotating priority rule does not place a restriction on when routers can send a probe. Any router can send out a *probe* msg. upon counter expiry. The priority rule is used to decide which *probe* can use a given output port in case of contention.

What if deadlock detection (and sending of probe) happen near the end of epoch? Suppose the epoch for this router starts at cycle 0 and the router sends a probe in ($3 \times t_{\mathsf{DD}}$ + k) cycle where 0 <k <t_{DD}. The next epoch for this router will begin at $4 \times$ N $\times t_{\mathsf{DD}}$ cycle where N is the

[5]Probe_Move is **not** forked unlike the probe msg. It traverses the deadlock path like the move SM.



(a) Two deadlock chains sharing routers.

(b) 8-ring Deadlock.

Fig. 5: **Examples of Special cases.**

number of routers in the network. Since the router sends a probe every t_{DD} cycle, the first probe in the next epoch will be sent in cycle 4 × N × t_{DD} + k. This probe has the highest priority for 3 × t_{DD} cycles. For appropriate choice of t_{DD} (such that the longest loop is covered)[6], we can guarantee that the probe msg. will be able to return to its sender without getting dropped due to contention.

It is not necessary, however, for the router to have the highest priority to solve deadlock. If its probe returns without facing contention from other probes in the network (common case), the recovery process can be started. Also, any router in the deadlock whose probe comes back can start the recovery.

What happens if two or more nodes in a deadlocked cycle detect and send out probes? A probe is dropped at a router if its dynamic priority is greater than the dynamic priority of the sender. Thus, only the probe from the node with highest dynamic priority in the deadlocked loop will be able to complete the loop.

Can a probe loop around infinitely due to buffer dependency? No, dynamically changing router priorities will make sure it gets dropped at some higher priority router. This however doesn't affect functional correctness as discussed before (principle of rotating priority and length of epoch).

Why do we need to fork the probe? Can we not drop the probe if all VCs at the input port do not want to use the same output port? There may be buffer dependency scenarios where one buffer dependency cycle may depend on another. In the walk-through example, if the probe message were to be dropped at node *4* and there was such a dependency cycle, the deadlock would never get resolved.

2) Correctness due to FSM: The FSM (Fig. 4(a)) handles all possible race conditions that might occur due to routers independently sending out SMs at various times. Here, in the interest of space, we discuss some interesting ones.

What if two deadlocked chains share routers? Multiple deadlocks can be resolved in parallel if they do not share routers. Otherwise they are resolved serially depending on which loop's *move* SM reached the common routers first. We show such a scenario in Fig. 5(a). There are 2 deadlock chains with routers K, M and E being the common routers. Router B and Router H, detect the anti-clockwise and clockwise deadlocks respectively and send out a probe.

Case I: Both probes arrive at the same cycle at the common router E from inports West and East respectively and request the same outport (North). The probe from router H is dropped at Router E as the dynamic priority of Router B (6) is greater than the dynamic priority of Router H (4) in the cycle at which probes reach Router E. Consequently, probe from Router B completes the loop and returns to its sender (Router B), which then sends out the *move SM* and performs the spin. The anti-clockwise deadlock gets resolved first in this scenario.

Case II: Both probes arrive at different cycles at the common router E. In this case, both probes will individually detect their respective cycles and return to routers H and B. Both routers will send out a *move* request. Whichever *move* request reaches router E first (say the one from H) will set the state to S_{Frozen}, latch the source_id of this *move* SM, and set $is\_deadlock$ to 1 in routers E, M, K, and so on. When the second *move* SM eventually arrives at E, it will be dropped since *is_deadlock* is set, and the *source-id* of this SM differs from what is latched, which is H. The clockwise deadlock through H will get resolved first in this case. Meanwhile, the FSM at router B will timeout without receiving its own *move* SM back, and it will send out *kill_move* SMs to defreeze routers C and D. *kill_move* will get dropped at E because of the source-id mismatch.

In either of these cases, once the first deadlock is resolved, either the second one will implicitly get resolved due to packet movement, or will explicitly lead to new probes and moves being sent out for resolution.

Folded/Non-Uniform cycles (*e.g.* Deadlock in a Figure "8") ? Such a scenario is depicted in Fig. 5(b). The design would view this as a single dependency chain (which it is).

Case I: Any router other than the cross-over router (Router E), say Router C, initiates the recovery by sending out a probe. When this probe reaches the South port of Router E, it will be sent out of the North port (not forked) since all flits in VCs at South port of Router E want to go North. Again, when this probe arrives at Router E from the East port, it will be sent out of the West port. A key difference from Fig. 5(a) is that the same *move* SM will circle back to router E twice, but will not be dropped the second time as its *source-id* is the same as the one that was latched the first time this *move* arrived.

Case II: The cross-over router, Router E, initiates the recovery by sending out a probe from the North outport. When this probe returns to Router E from the East port, it will not process it and send it out of the West outport since the probe didn't come back from the port the counter is pointing to (the South port). When the probe returns from the South port, the recovery process will proceed as discussed in Sec. IV-B.

In both the cases, Router E will move out both the frozen packets P1 and P2 from the North and West outports respectively in the SPIN cycle.

Can a node send a *probe*, followed by a *move*, and then receive a copy of its *probe* back? This means that there are

[6]For a 8x8 mesh, with 1-cycle link and 1-cycle router, t_{DD} of 128 is enough. Alternatively, the length of the epoch can be increased to n × t_{DD} (n>4) which would require lower t_{DD}

two dependence cycles that this router is part of, in the same direction. Since the *move* for the first one has been sent, the second *probe* will be dropped. Eventually the *move* will come back, and the spin will happen to resolve the first deadlock. After that the timeout counter will send out a new probe and resolve the second deadlock if it still exists.

3) Robustness: **Can the routers/links have different delays? Can it handle multiple clock domains in on/off-chip networks?** Yes. The theory only requires that all routers in the dependency chain start the SPIN together but not necessarily complete the SPIN at the same time. To calculate this common start time we only need the total loop delay (and not the delay of individual routers/links). Flits at routers with higher delays will take longer to show up at the downstream router. The space for incoming flits (due to SPIN) at the longer delay router is being created as flits are being moved out every cycle and the SPIN start time is same. If there is a different in rate due to difference in clock domain frequencies, cross-domain FIFOs are usually present between two such routers to smooth the flow/rate. In such designs, since the notion of global time is different, rather than specifying the exact cycle number in the *move* SM, the relative number of ticks after which the move has to be performed will be set. In off-chip systems, each SM (and the flits) will have to go through SERDES at each hop, which will just show up as increased loop delay without breaking the design.

D. Microarchitecture & Overheads

One of the key requirements for our implementation of SPIN to work is *deterministic traversal delays* for the SMs. The *probe* return time determines the length of the deadlock path; all FSM counter thresholds (except t_{DD} which is configurable) such as the *spin cycle* are derived directly from the length of the deadlock path. The *move* SM has to return within the the same time as the probe did, or triggers a *kill_move*, as discussed in Sec. IV-B5. Instead of using a separate dedicated side-band buffered network for these SMs, we provide these guarantees by the following features:

- **No additional links.** All SMs use the same links as regular flits and get higher priority over regular flits.
- **Bufferless Traversal.** The SMs are not buffered anywhere. Upon contention for the same output link, there is a strict priority order to decide which SM uses the output link (Sec. IV-C1), and all other SMs are dropped. The FSM which initiated the SM is robust to handle timeouts and retransmissions (Sec. IV-C2).
- **Distributed.** The recovery is completely distributed with no central coordinator limiting the timing.

Table II lists the additional modules required over a traditional on-chip/off-chip router. Our distributed SPIN router implementation adds one *Loop Buffer* (Sec. IV-B1) on the *control path*. We note, however, that we do not add any buffers on the datapath, making the loop buffer conceptually different from escape channels/buffers on the datapath [25], [6], [8], [26]. We quantify these overheads in Sec. VI-E3.

TABLE II: SPIN Router Modules

Module	Description
FSM	Manages SM traversals and Correctness (Figure 4(a) and Section IV-C2).
Probe Manager	Scans the VCs at the input port to find the set of unique output ports that flits in the VCs are waiting for and forks the received probe out of all of them.
Move Manager	Processes the *move*, *kill_move* and *probe_move* messages (Section IV-B) based on the FSM state.
Loop Buffer	Store the deadlock path and needs to be $\log_2$(Router_Radix) × N bits, where N is the no. of routers in the topology. For a 64-core mesh, this buffer would be 1-flit deep assuming 128-bit links.

V. The FAvORS Routing Algorithm

SPIN enables true one-VC fully adaptive deadlock free routing for any topology[7]. To exploit this powerful capability of SPIN, we propose a novel **F**ully **A**dapti**v**e **O**ne-VC **R**outing with **S**pin (FAvORS). The algorithm has two components : *minimal adaptive* and *non-minimal adaptive* routing.

Minimal Adaptive Routing: A packet is routed from the source node to the destination using only minimal paths. At a given router in the path, the router finds all the minimal paths to the destination. If one or more minimal paths have a free VC at the next hop router, the current router randomly chooses any of the minimal path output ports. If none of the minimal path outports have a free VC, the router selects the outport corresponding to the VC (at the next hop) that has been active for the least number of cycles (since the last time it became free). This information can be obtained cheaply from the VC credit. It is important to note here that a packet might switch to different minimal paths during routing due to outport selection but it is always routed minimally.

Non-minimal Adaptive Routing: Here the source router can choose between minimal adaptive routing and routing non-minimally through an intermediate node. The source finds all minimal paths to the destination which have a free-VC at the next hop and randomly selects one. If none of the minimal outports have a free-VC, the source node selects a random intermediate node. If H_{min} denotes the minimal path hop count, $H_{non-min}$ the non-minimal path hop count (from the source to destination via the intermediate node), $t_{active-min}$ ($t_{active-non-min}$) the minimum value of no. of cycles the next-hop VC has been active for[8], among all the minimal (non-minimal) paths, then the source router uses non-minimal routing via intermediate node if:

$$H_{min} + t_{active-min} > H_{non-min} + t_{active-non-min}$$

Else it would use minimal adaptive routing. The non-minimal routing has two phases : route to intermediate node from the source and then from intermediate node to the destination. In each of the phases, the packet is routed using the minimal adaptive routing algorithm described above. As the packet gets mis-routed only once, the routing is livelock free. The decision

[7]Traditional adaptive routing algorithms either require additional VCs for full adaptivity or routing restrictions, reducing adaptivity.

[8]If there is an idle VC then $t_{active-non-min}$=0. Also, the VC active time is relaxed by the buffer turn-around time.

to route minimally or non-minimally is only made once at the source router.

Algorithm Intuition: The scheme *favors* minimal routing due to the latency overhead of non-minimal routing. Non-minimal paths should only be considered in a heavily loaded network. If there exists a free-VC at the outport of the minimal path, it indicates that the network is lightly loaded and hence the algorithm routes minimally.

The algorithm uses "no. of cycles for which the outport VC has been active" as a proxy for port contention at the next router. If there is no contention, the VC at next hop router would return to the *idle* state fast. Further, to spread the traffic uniformly and prevent routing hotspots, the intermediate node is chosen randomly.

TABLE III: Network Configurations

Topology	Design	Adaptive	Minimal	Theory	Type
1024-node Dragonfly	UGAL	Full	No	Dally	Avoidance
	Minimal	No	Yes	SPIN	Recovery
	FAvORS_NMin	Full	No	SPIN	Recovery
8x8 2-D Mesh	Westfirst	Part	Yes	Dally	Avoidance
	EscapeVC	Full	Yes	Duato	Avoidance
	Static-Bubble [6]	Full	Yes	Flow-Ctrl	Recovery
	FAvORS_Min	Full	Yes	SPIN	Recovery

VI. Evaluation

A. Simulation Methodology

SPIN is a generic framework that can be used to provide freedom from routing deadlocks for any topology with any routing algorithm. We quantify its performance by comparing with state of the art deadlock free routing schemes on two popular topologies: a 1024 node off-chip dragon fly [16] with a group size of 8, and on-chip 8x8 mesh. All simulations are carried out using the cycle-accurate gem5 [38] full-system simulator with Garnet2.0 [39] network model. The mesh has 1-cycle routers and 1-cycle links; the dragon fly has 1-cycle routers [16], 1-cycle intra-group, and 3-cycle inter-group links. Both synthetic and real traffic is run over a directory coherence protocol that has 3 virtual networks to avoid protocol deadlocks; each vnet has one or more VCs as specified in the name of the configuration in each plot. For synthetic traffic, a mix of 1-flit (control) and 5-flit (data) packets were injected. The default value of t_{DD} is 128.

B. Baselines

Table III lists the state of the art baseline designs along with the type (Deadlock Avoidance/ Recovery) and theory they are based on. For off-chip 1024-node Dragon-fly topology, we use a deadlock avoidance scheme with UGAL routing that requires a change of VC every time a global link is traversed [16]. We quantify the performance of our FAvORS algorithm on Dragon-fly by comparing it with UGAL and minimal routing.

For 8x8 mesh, we choose a mix of avoidance and recovery schemes. WestFirst design uses west-first routing in all VCs while escape-VC design uses west-first routing in the escape VC and fully adaptive routing in rest. Static Bubble [6], a recently proposed deadlock recovery scheme for Mesh, uses adaptive routing in all VCs and strategically places additional buffers that are used for recovery. We note that the performance of other recovery schemes such as DISHA [25] would be similar to or worse than Static Bubble, as DISHA supports only one active recovery at a time besides having other overheads like token capturing, thus these techniques are not compared against directly. We also compare minimal version of FAvORS routing algorithm with all these baselines.

C. 1024-Node Off-Chip Dragon-Fly

Performance. Fig. 6 plots the latency vs. injection rate for dragon-fly topology for a suite of synthetic traffic patterns with a mix of 3-VC and 1-VC configurations from Table III.

In the 3VC version, we use the UGAL[37] routing algorithm, which is the most popular deadlock-free adaptive routing algorithm used commercially for dragon-fly topologies in HPC systems [40]. The baseline design with UGAL uses Dally's deadlock freedom theory and requires packets to change the VC on each use of the global/inter-group link to avoid routing deadlocks [16]. UGAL with SPIN allows packets to freely use any available VC. At very low loads (0.01) both designs perform identically. However, as SPIN provides more routing flexibility and lacks VC-use restrictions, its saturation throughput is 50%, 20% and 83% higher compared to the deadlock avoidance baseline with *bit_complement*, *transpose* and *tornado* traffic patterns[41] respectively. The effect of VC-use restriction on saturation throughput can be visualized better with the *neighbor* traffic pattern where router *i* sends all of its traffic to only router *i+1*. Here despite the presence of 3-VCs, the traffic in the baseline design can only use one-VC (for the minimal path which will be the dominant routing method for this traffic pattern) thereby severely limiting the throughput. Consequently, SPIN provides 25% improvement in saturation throughput.

In the 1-VC version, UGAL cannot be used as it requires at least 3 VCs. We run the fully-adaptive non-minimal version of the FAvORS algorithm, and contrast it against a minimal routing algorithm. The latter also uses SPIN for deadlock freedom[9]. FAvORS-NMin outperforms minimal routing by offering 78% and 62% higher throughput with *tornado* and *bit-complement* traffic patterns respectively. The higher throughput is achieved as FAvORS-NMin is able to adaptively route around loaded minimal paths. With *uniform random* traffic, FAvORS-NMin achieves a 5% improvement in saturation throughput as all links are almost equally loaded. With *transpose* and *neighbor* traffic patterns, FAvORS-NMin always picks the minimal path (see Sec. V) and thus the performance of both designs is identical.

Performance/Watt and Performance/Area. A key take-away from Fig. 6 is that FAvORS provides the same low-load latency and similar throughput as UGAL with 3VCs. In terms of cost, the 1-VC dragon-fly router is 53% lower area and 55% lower power than a 3-VC version, when implemented in Nangate 15nm opencell library [42].

[9]No other truly one-VC deadlock-free routing designs exist. State of the art deadlock-free routing schemes for dragon-fly require a minimum of 2 VCs.

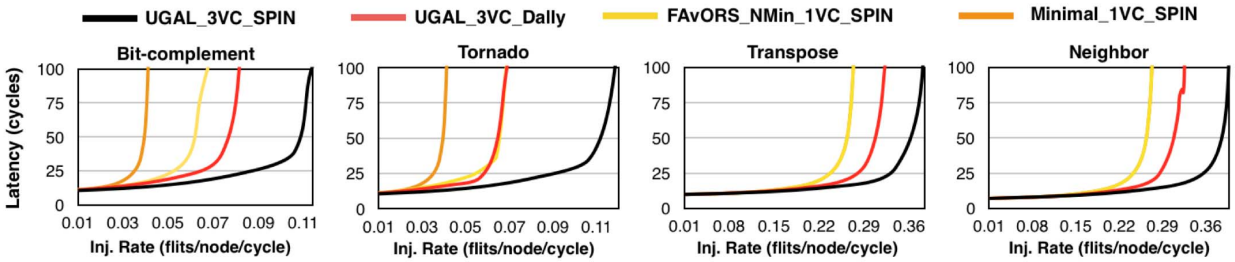


Fig. 6: **Network Performance of 1024-node Dragon Fly (configs described in Table III).**

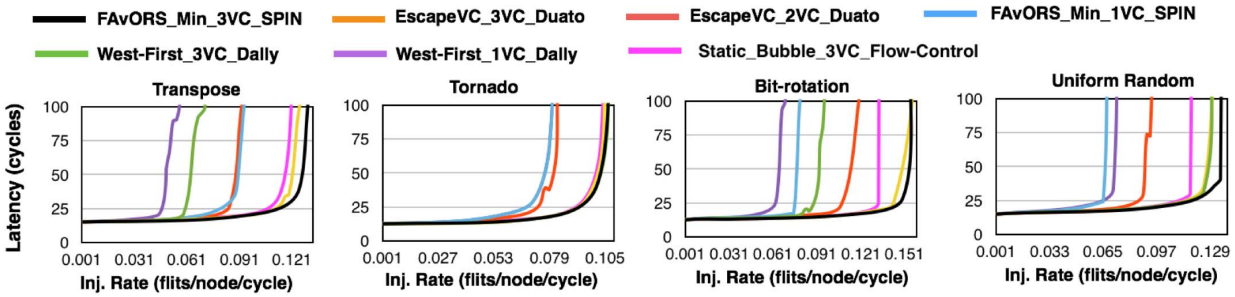


Fig. 7: **Network Performance of 8x8 2-D Mesh (configs described in Table III)**

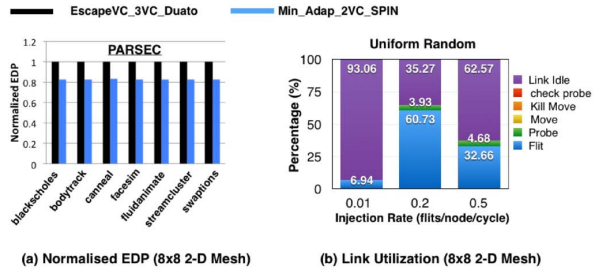


Fig. 8: **Normalized network EDP with PARSEC and Link Utilization with Uniform Random Traffic for 2-D Mesh.**

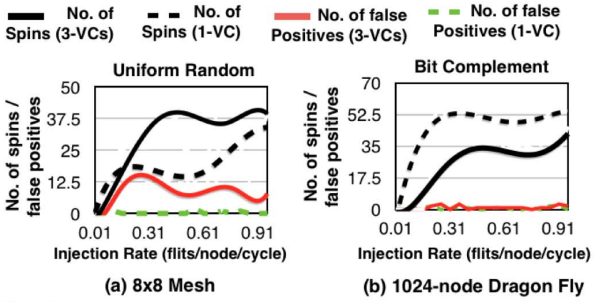


Fig. 9: **False positives in SPIN as a function of number of VCs in 2-D Mesh and Dragon Fly.**

In summary, SPIN enables high-performance deadlock-free routing at almost half the area and power cost of conventional deadlock avoidance techniques in high-radix HPC topologies like Dragon-Fly.

D. 8x8 On-Chip 2-D Mesh.

Performance. Fig. 7 plots the latency versus injection rate for a 8x8 mesh for a suite of synthetic traffic patterns with a mix of VC configurations from Table III.

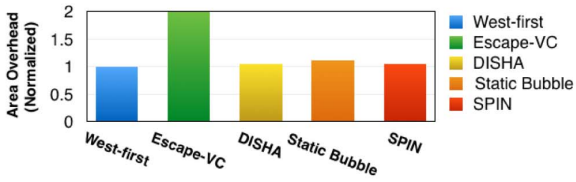


Fig. 10: **Area overhead of baseline designs with the SPIN implementation normalized to the West-first (deadlock avoidance) design.**

In the 3-VC designs, minimal adaptive with SPIN outperforms the other 3 VC baselines offering higher saturation throughput as it provides adaptive routing for all 3 VCs and does not place any routing restriction (west-first) or VC use restriction (escape-VC). Quantitatively, SPIN provides 79%, 16% and 68% higher saturation throughput than west-first routing baseline and 6%, 18% and and 8% improvement in saturation throughput compared to the escape-vc design with *bit reverse*, *uniform random* and *transpose* traffic patterns respectively. With *tornado* traffic pattern, each router sends all of its traffic to a node half-way across the x-dimension. Minimal adaptive routing reduces to west-first routing for this traffic pattern for most packets and thus the performance of all three designs is almost identical. Compared to the Static Bubble flow-control based deadlock recovery scheme, SPIN provides 10% and 14% higher throughput for *transpose* and *bit rotation* respectively. The primary reason is that one of the VCs in Static Bubble is reserved for deadlock recovery and cannot be used during normal operation.

In the 1-VC designs, FAvORS-Min provides 80%, 20% and 18% higher saturation throughput with *transpose*, *bit-reverse* and *bit-rotation* traffic patterns respectively. With *neighbor*

(not shown) and *tornado* traffic patterns minimal adaptive routing reduces to west-first routing; thus both designs offer identical performance. West-first routing shows a marginal improvement (3%) in saturation throughput over the adaptive routing algorithm with *uniform random* traffic.

Performance/Watt and Performance/Area. Fully-adaptive FAvORS-Min with 1-VC enabled by SPIN matches in throughput with EscapeVC_2VC for *transpose* and *tornado*, and is only 10% worse in throughput than a 3-VC West-first design for *bit_rotation*. Since Escape VC offers comparable performance to SPIN for on-chip meshes, we compared the two with full-system simulations with the PARSEC [43] benchmark suite. Fig. 8(a) plots the network EDP, normalized to Escape VC. MinAdaptive_2VC_SPIN has a 18% lower EDP, on average, compared to EscapeVC_3VC as it provides the same performance with fewer resources. Full system runtime was identical with both the designs.

The 1-VC mesh router is 52% (36%) lower area and 50% (34%) lower power than a 3-VC (2-VC) router when implemented in Nangate 15nm opencell library [42].

In summary, SPIN provides better or similar performance at 30-50% lower area and power costs for on-chip mesh. Moreover, since SPIN can handle any arbitrary topology, it has high-applicability in the domain of on-chip resiliency and NoC power gating.

E. Accuracy and Overheads of SPIN

1) False Positives: Deadlock resolution in SPIN is carried out without requiring the routers to have a global view of the network which contributes to the scheme's scalability. However due to the absence of a global view, routers may send out SMs due to congestion when there is no real deadlock. These are known as false positives. As discussed in Sec. IV-C routers are robust against such false positives and drop them. Fig. 9 plots the number of *false positives* and number of *spins* as a function of injection rate with *uniform random* and *bit-complement*[10] patterns for mesh and dragon-fly respectively.

For the mesh topology, 20% of the spins on average are false positives for the 3-VC design. However, the number of false positives are zero up to 10x the injection rate of real applications [43], [6]. For the 1-VC case, there are no false positives for any injection rate. In the case of dragon-fly topology, the false positives are very close to zero for both 1-VC and 3-VC designs at all injection rates. An interesting insight offered by this graph is that the number of false positives fall significantly in the 1-VC design. This is because in the multi-VC scenario, VCs at the input port of the router may be part of different inter-dependent dependency chains. A *probe* SM that arrives at such a router will be forked out of multiple output ports. Often, clearing one of the chains allows the others to make forward progress as well. Consequently, all copies of the *probe* SM (except one) get dropped. In the 1-VC design, probe-forking cannot happen and consequently the *false* positives are zero.

[10]Uniform random traffic did not deadlock for dragon-fly topology with 3-VCs (even at high injection rate) due to the abundant path diversity.

The probability of deadlocks decreases with more VCs [8]. This is shown by the number of *spins* for 1-VC and 3-VC design with dragon-fly topology where the 3-VC design has 43% less *spins* on average than the 1-VC design. The same trend is reflected for mesh topology at low and medium loads (up to 0.2). At high loads, more *spins* are observed for the 3-VC design. This is explained by the difference in routing choices available in the two designs. The 3-VC case allows for better exploitation of path diversity. Consequently, the packets may take any turns leading to higher probability of deadlock. In the 1-VC design, packets are unable to exploit the path diversity in a heavily saturated VC-limited network. Thus, their paths contain less turns (like XY routing) leading to lower probability of deadlock.

2) Link Utilization: Fig. 8(b) plots the link utilization for flits and special msgs. and the link idle time with uniform random traffic for the mesh topology with 3-VCs at three different injection rates : 0.01 (low load), 0.2 (medium load) and 0.5 (high load). The design uses minimal adaptive routing with SPIN. At low load, links are mostly idle and no special msgs. are sent out. At medium load, the flit link utilization goes up to 60% with the *probe* utilization being around 4%. The *probe* utilization remains almost the same at high load with the flit utilization dropping to 33% due to increased frequency of deadlocks at high load. It is interesting to note the increase in link idle time in the high load case indicating that the links are mostly idle in case of frequent deadlocks and can thus be used by the SM. All SMs have less than 1% link utilization combined at both medium and high loads which indicates that the *spin* is carried out only in the case of a deadlock. The combined link utilization of SMs never exceeds 5% at any load; thus the links are either idle or being used by flits at almost all times.

3) Area Overhead: We implemented the SPIN modules (Table II) in RTL over a 1-cycle router obtained from a NoC RTL generator [13], and synthesized the design using 15nm Nangate standard cells [42]. The area overheads compared to alternate designs are shown in Figure 10. SPIN has 4% area overhead compared to the West-first scheme which is significantly less than 100% and 10% area overhead for Escape-VC [8] and Static Bubble [6] designs respectively. We believe that a 4% area overhead is a highly acceptable trade-off for the routing flexibility and higher saturation throughput that SPIN provides compared to Dally-based deadlock avoidance schemes in use today. SPIN has similar area overheads as other deadlock recovery schemes, such as DISHA [25] and Static Bubble [6] which also use an additional central buffer in each router, but provides better performance as discussed before.

VII. Conclusion

This work presents a novel deadlock freedom framework called SPIN. SPIN views deadlocks as a lack of coordination between routers in a cyclic dependence loop, making it fundamentally different from all prior theories that have viewed deadlocks as a lack of buffer resource problem thereby

requiring routing restrictions or additional buffers or conservative injection restrictions. We also present a low-cost, fully-distributed microarchitecture for detecting a deadlock and effectuating a one (or more) hop movement of a deadlocked ring in *any topology*. We also enable, for the first time, a fully adaptive routing algorithm that can work with just 1 VC by leveraging SPIN. Evaluations over a 1024-node off-chip Dragon-Fly and a 64-node on-chip mesh demonstrate that SPIN can provide comparable performance to state-of-the-art deadlock-free routing algorithms at 35-52% lower area and 38-55% lower power.

REFERENCES

[1] A. Singh *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), ACM, 2015.

[2] A. Roy *et al.*, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), ACM, 2015.

[3] D. Chen *et al.*, "The ibm blue gene/q interconnection network and message unit," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, (New York, NY, USA), ACM, 2011.

[4] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, Sept. 2007.

[5] C. J. Glass *et al.*, "The turn model for adaptive routing," *J. ACM*, vol. 41, Sept. 1994.

[6] 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.

[7] W. J. Dally *et al.*, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, May 1987.

[8] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, Dec. 1993.

[9] C. Carrion *et al.*, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proceedings of the Fourth International Conference on High-Performance Computing*, HIPC '97, (Washington, DC, USA), IEEE Computer Society, 1997.

[10] T. Moscibroda *et al.*, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), ACM, 2009.

[11] A. Singla *et al.*, "Jellyfish: Networking data centers randomly," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), USENIX Association, 2012.

[12] L. Chen *et al.*, "Power punch: Towards non-blocking power-gating of noc routers," in *21st IEEE International Symposium on High Performance Computer Architecture, HPCA*, pp. 378–389, 2015.

[13] H. Kwon and T. Krishna, "Opensmart: Single-cycle multi-hop noc generator in bsv and chisel," in *Proc of the IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE, 2017.

[14] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, 2017.

[15] Y.-H. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, IEEE, 2016.

[16] J. Kim *et al.*, "Technology-driven, highly-scalable dragonfly topology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, (Washington, DC, USA), IEEE Computer Society, 2008.

[17] M. Ebrahimi *et al.*, "A new theory on forming acyclic channel dependency graph for the design of deadlock-free networks," in *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, 2017.

[18] C. Xiao *et al.*, "Dimensional bubble flow control and fully adaptive routing in the 2-d mesh network on chip," in *2008 IEEE/IPIP International Conference on Embedded and Ubiquitous Computing*, 2008.

[19] M. Garcia *et al.*, "On-the-fly adaptive routing in high-radix hierarchical networks," in *Proceedings of the 2012 41st International Conference on Parallel Processing*, ICPP '12, (Washington, DC, USA), IEEE Computer Society, 2012.

[20] K. Aisopos *et al.*, "ARIADNE: agnostic reconfiguration in a disconnected network environment," in *International Conference on Parallel Architectures and Compilation Techniques, PACT*, 2011.

[21] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, July 2000.

[22] W. J. Dally *et al.*, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, Apr. 1993.

[23] P. Gratz *et al.*, "Regional congestion awareness for load balance in networks-on-chip," in *14th International Conference on High-Performance Computer Architecture (HPCA-14 2008),*, 2008.

[24] S. Ma *et al.*, "Dbar: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, (New York, NY, USA), ACM, 2011.

[25] K. V. Anjan *et al.*, "An efficient, fully adaptive deadlock recovery scheme: DISHA," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture, ISCA*, 1995.

[26] Y. H. Song *et al.*, "A new mechanism for congestion and deadlock resolution," in *Proceedings of the 2002 International Conference on Parallel Processing*, ICPP '02, (Washington, DC, USA), IEEE Computer Society, 2002.

[27] P. Lopez *et al.*, "A very efficient distributed deadlock detection mechanism for wormhole networks," in *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, 1998.

[28] J. Duato *et al.*, "A general theory for deadlock-free adaptive routing using a mixed set of resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, Dec. 2001.

[29] A. Samih *et al.*, "Energy-efficient interconnect via router parking," in *19th IEEE International Symposium on High Performance Computer Architecture, HPCA*, 2013.

[30] V. Puente *et al.*, "The adaptive bubble router," *J. Parallel Distrib. Comput.*, vol. 61, Sept. 2001.

[31] L. Chen *et al.*, "Worm-bubble flow control," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, (Washington, DC, USA), IEEE Computer Society, 2013.

[32] L. Chen *et al.*, "Critical bubble scheme: An efficient implementation of globally aware network flow control," in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011,*, 2011.

[33] C. Xiao *et al.*, "On distributed communication networks," in *1964 IEEE Trans. on Communications*.

[34] C. Fallin *et al.*, "Minbd: Minimally-buffered deflection routing for energy-efficient interconnect," in *2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NoCS),*, 2012.

[35] C. Fallin *et al.*, "Chipper: A low-complexity bufferless deflection router," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, (Washington, DC, USA), IEEE Computer Society, 2011.

[36] S. A. R. Jafri *et al.*, "Adaptive flow control for robust performance and energy," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, (Washington, DC, USA), IEEE Computer Society, 2010.

[37] A. Singh, *Load-balanced routing in interconnection networks*. PhD thesis, Stanford University, 2005.

[38] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, Aug. 2011.

[39] N. Agarwal *et al.*, "GARNET: A detailed on-chip network model inside a full-system simulator," in *ISPASS*, 2009.

[40] "Cray XC Series Interconnect and Network Technology," Cray.

[41] "Garnet synthetic traffic," gem5.org.

[42] M. Martins *et al.*, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ACM, 2015.

[43] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.