BINDU: Deadlock-Freedom with One Bubble in the Network

Mayank Parasar **Tushar Krishna**
School of ECE, Georgia Institute of Technology, Atlanta, GA, USA
mparasar3@gatech.edu *tushar@ece.gatech.edu*

ABSTRACT

Every interconnection network must ensure, for its functional correctness, that it is deadlock free. A routing deadlock occurs when there is a cyclic dependency of packets when acquiring the buffers of the routers. Prior solutions have provisioned an extra set of *escape* buffers to resolve deadlocks, or restrict the path that a packet can take in the network by disallowing certain turns. This either pays higher power/area overhead or impacts performance. In this work, we demonstrate that (i) keeping one virtual-channel in the entire network (called 'Bindu') empty, and (ii) forcing it to move through all input ports of every router in the network via a pre-defined path, can guarantee deadlock-freedom. We show that our scheme (a) is topology agnostic (we evaluate it on multiple topologies, both regular and irregular), (b) does not impose any turn restrictions on packets, (c) does not require an extra set of escape buffers, and (d) is free from the complex circuitry for detecting and recovering from deadlocks. We report 15% average improvement in throughput for synthetic traffic and 7% average reduction in runtime for real applications over state-of-the-art deadlock freedom schemes.

KEYWORDS

Computer architecture, Network-on-chip, Interconnection network, Deadlock

ACM Reference Format:
Mayank Parasar **Tushar Krishna**. 2019. BINDU: Deadlock-Freedom with One Bubble in the Network . In *International Symposium on Networks-on-Chip (NOCS '19), October 17–18, 2019, New York, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3313231.3352359

1 INTRODUCTION

Deadlock-freedom is necessary for designing any functionally correct interconnection network. Routing-level deadlocks occur when there is a cyclic dependency between the packets in the network as they try to acquire the buffers. Prior solutions on routing deadlocks can mainly be categorized into *deadlock avoidance* and *deadlock recovery*.

Deadlock avoidance [9, 12, 19] designs the routing algorithm such that a cyclic dependence between packets can never get created at runtime to begin with by disabling certain turns. These *turn models* are extremely popular in regular network topologies such as Mesh - classic examples being XY, West-first and their variants [12]. These turn models can be implemented for all packets or only those within certain *escape* virtual channels (VCs) [11]. Deadlock avoidance-based solutions become challenging to adopt for irregular topologies (which may occur due to heterogeneous SoCs, network faults, or power-gating) [16] as they require complex analysis of the channel dependence graph (CDG) to disable turns, and also lead to performance losses due to non-minimal routes [16].

In deadlock-recovery [3, 16, 17], deadlocks are dynamically detected and resolved. These solutions are amenable to arbitrary topologies [16, 17]; however, they are extremely complicated to implement due to global deadlock-detection and resolution.

There is another class of solutions that achieves deadlock freedom using *bubbles* (i.e. empty VCs or buffers) in the network. These techniques allow for cycles in the CDG but control injection of packets such that packets never get into a cyclic buffer dependency at runtime [6, 7, 15]. The underlying theory, Bubble Flow Control [15], states that one bubble within a cyclic dependence can ensure forward progress. Multiple implementations of this theory exist today. Critical Bubble Scheme (CBS) [7] and its variants [6] are the state-of-the-art for rings and Torii; here one bubble in every ring is marked as a *critical bubble* and circulates through the ring. Static Bubble (SB) [16] embeds extra buffers at various routers at design time, and turns them ON upon deadlock-detection to introduce a bubble. Brownian Bubble Router (BBR) [14] reserves one bubble in every router in the network, and circulates it across the router's ports. CBS, SB and BBR require $4k$, $O(k)$ [16], and k^2 bubbles respectively in a $k \times k$ mesh/torus[1]. More bubbles naturally lead to reduced throughput and low buffer utilization.

In this work, called BINDU (Bubble in Irregular Network for Deadlock pUrging), we demonstrate, for the first time, that it is possible to provide deadlock freedom with just a *single* bubble (referred to as a 'Bindu'[2] and defined formally in §3.1) in the *entire network*. The Bindu moves through the network in a fixed path, covering all the routers and their input ports in the network. During its course the Bindu shuffles the packets present in the network, naturally resolving any deadlock that comes in its path.

The following are the primary contributions of this work:

(1) A novel technique to guarantee deadlock freedom in arbitrary irregular topologies by having *one* or more Bindus (empty VC) in the entire network, and to force these Bindus to move through-out the network at a periodic rate.
(2) BINDU frees the designer from any consideration of deadlock when designing their routing algorithm, allowing high performance with minimal overhead.
(3) And evaluation of how BINDU performance compares with previously proposed deadlock freedom techniques with both

NOCS '19, October 17–18, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6700-4/19/10…$15.00
https://doi.org/10.1145/3313231.3352359

[1] In practice, the number of bubbles needed during operation in CBS becomes higher since transitioning from one ring to the other requires at least two bubbles in that ring.
[2] Bindu is a Hindi word, meaning 'point'. With a collection of points, we can draw any geometrical figure; similarly with BINDU we can make any topology deadlock free.

synthetic traffic and real applications on both regular and irregular topologies[3].

The rest of the paper is organized as follows: §2 provides background information and related work on deadlock-freedom in NoCs; §3 deep dives into the BINDU technique, discussing the concept, proof of deadlock-freedom, and implementation; §4 then delves into experimental results, and §5 presents conclusion.

2 BACKGROUND AND RELATED WORK

Deadlocks refer to a condition in which a set of agents (i.e. packets) wait indefinitely trying to acquire a set of resources (input buffers). Deadlocks primarily occur because of the cycles present in the Channel Dependency Graph (CDG) of the topology. The CDG is a directed graph, in which each node is a link in the topology and each edge in CDG defines the order in which packets want to traverse the topology to reach its destination. Since there could be multiple paths from a given source to destination, there could be many edges in CDG connecting two different nodes. These edges can form a cycle in themselves leading to a deadlock. The choices of the path that a packet can take are determined by the routing algorithm, therefore CDG is the function of both routing algorithm and topology. We classify deadlock resolution solutions in NoCs into the following categories: deadlock-avoidance, deadlock-recovery, and bubble-based[4]. A High-level qualitative comparison of BINDU with state-of-the-art schemes have been paraphrased in Table 1.

2.0.1 Deadlock Avoidance. In deadlock avoidance [8, 9, 19] schemes, the CDG is made acyclic via turn-restrictions in the routing algorithm (e..g, XY, West-first, and so on in a Mesh, Up-Down [18] routing in irregular topologies), ensuring that a cyclic dependence is never created during runtime. However, the loss in path-diversity leads to loss in network throughput. To address this, escape VC [11] based schemes are often used, where the turn restrictions are present in only a subset of VCs, while the rest of the VCs can use all paths. The CDG can therefore have cycles, but there is at least one acyclic escape path to which all packets have access. Escape VC based schemes however lead to an increase in the total number of VCs, adding area and power at every router. Another key challenge with the aforementioned deadlock avoidance schemes, beyond the performance or area/power implications, is that they are highly topology-dependent, since every topology has a unique CDG. As topologies become irregular (due to heterogeneity, or waning silicon reliability or due to power gating of routers/links), coming up with unique turn restrictions and encoding them in the packets is an expensive operation.

2.0.2 Deadlock Recovery. Another school of thought for resolving deadlocks stems from the observation that deadlocks are a rare phenomenon, therefore instead of adding turn restrictions or extra VCs in the routers to avoid deadlocks, one should detect, locate, and resolve deadlocks. Deadlock recovery algorithms allow packets to take all paths provided by the topology to reach to their destination. However, in doing so, the packets can get deadlocked due to cyclic CDGs. To recover from it, these techniques require detection via timeouts, location via probes, and resolution via synchronization messages [2, 3, 16, 17]. Implementing deadlock recovery is therefore quite complex. Moreover, this technique is not scalable; as network sizes increase, the probability, shape, and length of deadlock-ring also increases, making it even more difficult to locate them dynamically [16]. There have also been proposals to drop flits in the network [23], if they fail to win the switch for a threshold number of tries, thereby breaking any deadlocks. However, this approach comes with the overhead of tracking and re-transmitting the dropped flit as NoCs do not tolerate packet/flit losses.

2.0.3 Bubble based approaches. Bubble flow control (BFC) [15] based approaches fall in between the avoidance and recovery based solutions presented above. They allow cyclic CDGs - but guarantee that cycles will actually not get created at runtime by ensuring that at least one bubble (i.e., empty VC [7, 16] or flit-buffer [6]) will be present within any cycle, to provide forward progress. Thus the most popular implementations of BFCs is in rings and torii [6, 7, 15]. Challenge with BFC and its variants is that each ring in the topology

Table 1: Qualitative Comparison of Deadlock Freedom Mechanisms

Mechanism	Full Path Diversity	High Throughput	No Extra VCs	No D'lock Detect	Topology Agnostic
CDG [9]	✗	✗	✓	✓	✗
Escape VC [10]	✓/✗**	✓	✗	✓	✓/✗**
BFC [7, 15]	✓	✗	✓	✓	✗
Deflection [13]	✓/✗*	✗	✓	✓	✓
Recovery [16, 17]	✓	✓	✓	✗	✓
BBR [14]	✓	✗	✓	✓	✓
BINDU	✓	✓	✓	✓	✓

*At low-loads, full path diversity is available. But at high loads, packets cannot control the directions or paths along with they are deflected.
** Within esc VC: limited path diversity + requires topology info to drain.

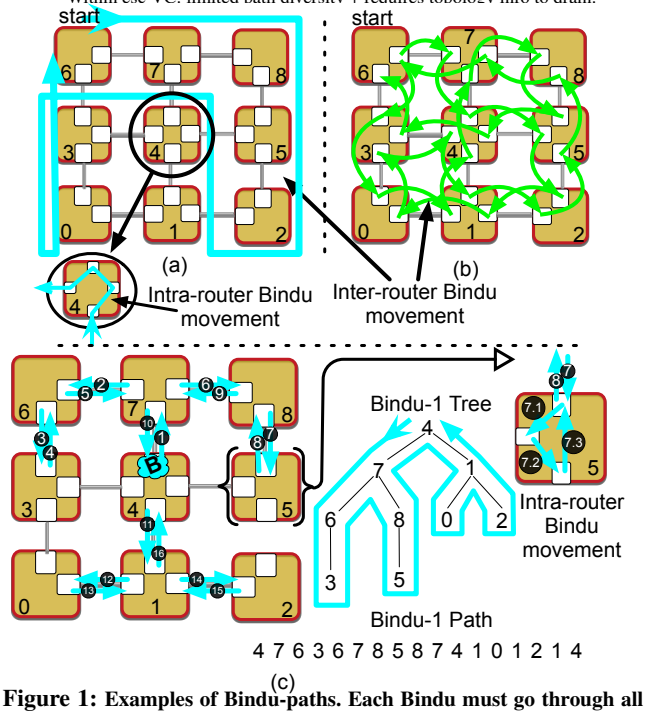

Figure 1: Examples of Bindu-paths. Each Bindu must go through all input ports of all routers of the network, at least once. (a) Bindu moving through all ports of a router before jumping to the next router, (b) Bindu jumping between input ports of different routers throughout its path, (c) A tree-based Bindu-path for an irregular topology

[3]We would like to note that the focus of this work is not on the important problem of dynamic fault-tolerance; rather it guarantees deadlock freedom in static irregular topologies, which may be created due to faults or at design time.

[4]In the literature, bubble-based schemes are often classified as flow-control based schemes within deadlock avoidance [17]. We classify them separately to allow us to take an in-depth view and compare them against BINDU.

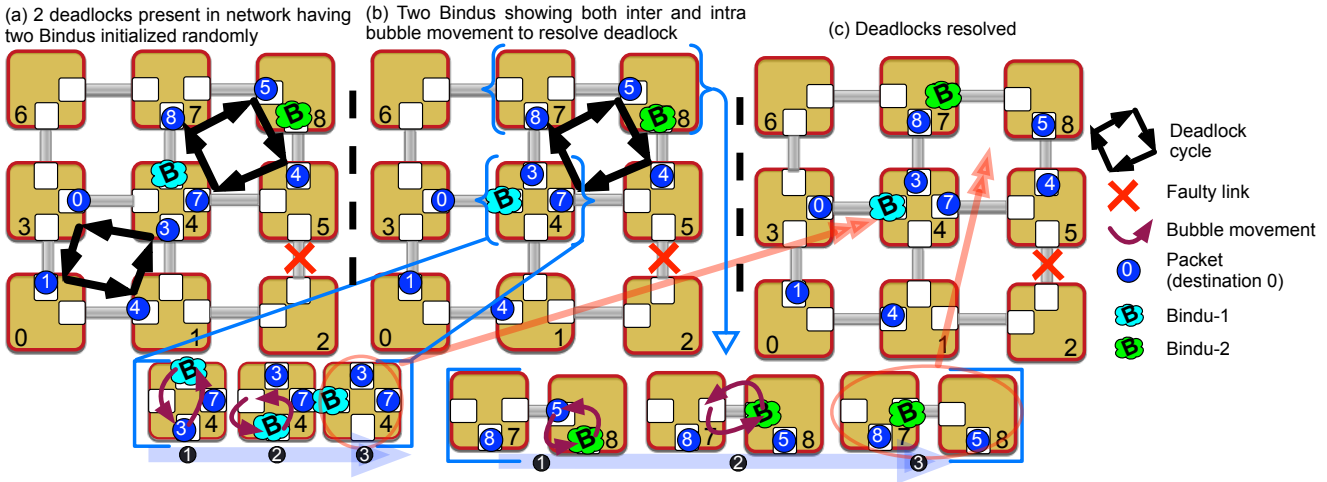

Figure 2: Walkthrough figure showing the BINDU in action. Here deadlock involving router-0,1,3 and 4 is resolved by intra-router Bindu movement of Bindu-1 and deadlock involving router-4, 5, 7 and 8 is resolved by inter-router Bindu movement of Bindu-2. Network state corresponding to each type of Bindu movement is shown in sub-figure (b) and (c) respectively.

needs at least one bubble (and two for injection), leading to a loss in throughput. BFC has been implemented in arbitrary topologies [22], two such example of prior works are as follows. Static Bubble [16] is a recovery based scheme where a subset of routers have extra buffers inserted at design time that are 'off'. Upon location of the deadlocked ring, the buffer in one of the routers is turned on to introduce a bubble into the ring. However, this scheme requires expensive circuitry for deadlock detection and turning the bubble on and off. BBR [14] keeps one bubble in every router, and keeps it circulating through all the input ports of the router to introduce bubbles in deadlocks going through that router. However, because of the significant number of bubbles in the network (one per router) this scheme suffers from low throughput.

In this paper, we present a deadlock-freedom solution using a single bubble (called Bindu) in the entire network.

3 BINDU NETWORK

3.1 Definitions

We formally define terms here that we will use throughout the paper.

Bindu: In BINDU, *Bindu* refers to a reserved packet-sized *empty VC*, that is instantiated at the starting of the network run. It makes pro-active movement throughout the network covering all the input ports of every router in the network, as shown in Fig. 1. Unlike 'Bubble' used in previous works [7, 15], no packet can sit inside the Bindu. As a Bindu procatively moves through the network, packets get displaced, as shown in the walk-through Fig. 2.

k-Bindu: BINDU networks can incorporate multiple Bindus within the network, each following its own path as shown in Fig. 1(c) and Fig. 2. We refer to this configuration as 'k-Bindu', where 'k' is the number of Bindus instantiated at the starting of network run.

Bindu movement: Movement of Bindu from one VC to another within the router or across routers. Moving a Bindu from <Router_i, Port_m, VC_k> to <Router_i+1 Port_n, VC_o>, effectively means reading the packet from <..., VC_o> and writing it to <..., VC_k>[5].

[5]We assume Virtual Cut-Through, i.e., all VCs that the Bindu traverses are sized to hold the largest packet. Wormhole designs will be discussed in §3.6

Blocked Packet: A packet which is *indefinitely stuck* because it is part of a deadlock ring.

Unblocked Packet: A packet which might be *temporarily stalled*, because of congestion in the network or unavailability of credits at the downstream router. However, it is *guaranteed to eventually leave the router*.

Empty slot: An empty VC in a router.

3.2 Basic Idea and Walk-through Example

Deadlocks are characterized by cyclic dependency of packets, which renders the forward movement impossible. BINDU tries to resolve this cyclic dependency using one or more Bindus in the network. A Bindu can be randomly initialized at one of input VCs of any router, barring the injection input VCs. Multiple Bindus can co-exist within a router - but cannot reserve the same VC at the same time.

Each Bindu pro-actively moves in a predefined path which cycles back from where it started (Fig. 1). Bindu's movement results in the partial shuffling of packets in the network. This results in naturally resolving any deadlock cycle which comes in Bindu's path. To understand the BINDU technique in more detail, let us walk-through the scheme with an example. Fig. 2(a) shows two deadlocks in a 3x3 Mesh; the link between router-2 and router-5 is faulty, resulting in an irregular topology. The number on the packet refers to its destination router-id. Even though one Bindu is enough to resolve the deadlock, we show two Bindus in the walk-through figure to underline the generality of the scheme.

In Fig. 2(b), we focus on the deadlock between packets in Routers-0, 1, 3 and 4. Bindu-1 moves within Router-4 from the North to the South to the West port. This results in an empty slot in the South port, which can now be used by the packet stuck in Router-1 to make forward progress, resolving the deadlock.

In Fig. 2(c), we focus on the deadlock between packets in Routers-4, 5, 7 and 8. Bindu-2 jumps from Router-8 to Router-7. An important point to observe is that Bindu-2 first traverses within Router-8 from South to West (similar to Bindu-1), and then jumps to the connected Router-7 to its East port. The deadlock is resolved as Bindu

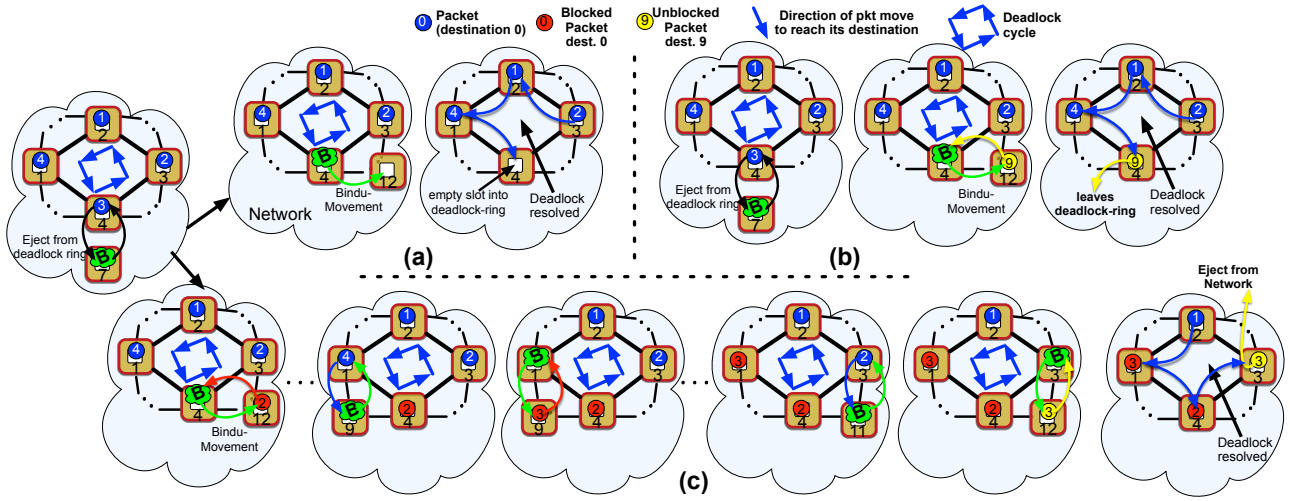

Figure 3: The figure shows: (a) The way Bindu resolves the deadlock when it brings an empty slot to the deadlock ring. (b) How Bindu resolves the deadlock when it brings a unblocked packet to the deadlock ring. (c) Bindu resolves the deadlock by shuffling the packets present within the deadlock ring. The number inside the packet refers to its destination.

replaces an earlier blocked (i.e., deadlocked) packet at West input port of Router-8 with an empty VC.

All Bindus need to traverse through all ports of all routers, as we discuss in §3.3. The implications of Bindu's intra and inter-router movements on the router micro-architecture are discussed in §3.6.

3.3 Bindu Path

All Bindus move through *all the input ports of every router* in the topology (both regular and irregular) at a periodic rate indefinitely. There can be multiple possible paths. Fig. 1 shows three possible paths for Bindus in a 3×3 Mesh. In Fig. 1(a), each Bindu snakes through the network routers, moving through all input ports at each router; in Fig. 1(b), Bindus jump between input ports of different routers in each step; Fig. 1(c), the Bindu uses a tree to loop through the entire network as it is an irregular topology where the original snake does not work. As mentioned earlier, there can be more than one Bindu in the network, and each Bindu can choose different paths. The Bindu-path is encoded within each router; i.e., each router has a preset order of input ports through which the Bindu should move, and a preset neighbor to which the Bindu should jump. The next inport-id (intra-router) or router-id (inter-router) where Bindu needs to move is provided by the current router. A router can have multiple paths encoded, indexed by the Bindu id. We discuss static vs. dynamic configurability of Bindu paths in §3.6.

3.4 Bindu Movement

Moving a Bindu from VC-A to VC-B requires explicitly moving the packet from VC-B to VC-A. VC-B could be a VC within the router (during intra-router Bindu movement), or at a neighboring router (during inter-router Bindu movement). Moving a Bindu across routers can lead to a temporary misroute of the packet.

In our design, we constrain Bindu to move only across VC-0 within the input ports of all routers in the network during both intra- and inter-router Bindu movement. This decision simplifies the router micro-architecture for BINDU (§3.6). VC-0 is Virtual-Cut Through. A Bindu movement takes f cycles, where f is the number of flits in the packet. Naturally, the Bindu movement period p needs to be greater than the size of the largest packet that can sit in VC-0.

3.5 Proof of Deadlock freedom

In this section, we explain the proof of BINDU technique for deadlock freedom using Fig. 3 as reference. We refer to the terms defined in §3.1 and provide the following arguments.

(1) By the virtue of the Bindu's looped path, it is guaranteed to visit every deadlock ring in the network.
(2) As Bindu moves into the deadlock ring and then moves out of deadlock ring it either brings a fresh packet or an empty slot into the deadlock ring.
(3) If Bindu brings in the empty slot at its place (Fig. 3-(a)), then, by default, it resolves the deadlock as the earlier deadlocked packet can take up that empty slot breaking the deadlock dependency.
(4) If Bindu brings in a fresh packet to the deadlock ring, then there could be two possibilities. This fresh packet can either be 'unblocked' or 'blocked', as defined in §3.1.
(5) If the fresh packet is unblocked (Fig. 3-(b)), it will naturally leave the deadlock ring; this will create an empty slot in the deadlock ring, and deadlock will naturally get resolved as described in point 3.
(6) However, if the fresh packet is blocked, then deadlock would persist until the next Bindu movement into the deadlock ring. If the next Bindu movement creates the empty slot or brings in unblocked fresh packet to deadlock ring, then deadlock will get resolved as mentioned in point 3 and point 5.
(7) There could be a very rare, corner case in which Bindu movement will keep bringing a blocked packet to the deadlock ring (Fig. 3-(c)). This, in fact, means that the packets being brought into the deadlock ring by Bindu are part of the same deadlock ring. Here, Bindu movement effectively shuffles the packets within the deadlock ring. This shuffling of packets within the deadlock ring ensures that eventually, at least one packet would reach to its destination because of shuffling and eject-out of the network as shown in Fig. 3-(c). This would finally lead to deadlock resolution.

(8) A final pathological corner case could be when all packets of the network are in one big deadlock loop (e.g., this could occur in a ring topology). In such a scenario, the movement of packets due to Bindu will continue to remain in a cyclic loop. However, once a Bindu completes a full looped path, all packets would have effectively been spun around, as described in [17]. Eventually after 'k' spins one of the packets would reach its destination and eject out from the network. This would again lead to deadlock resolution.

It is worth noting that we did not actually observe either (7) or (8) in our extensive experiments.

Livelocks: Livelock is the condition where a packet keeps moving indefinitely but *never* reaches its destination. Recall that packets get misrouted by one-hop due to a Bindu movement. As long as the packet makes two forward hops before a Bindu misroutes it again, it will not livelock. Since Bindu paths are fixed, for the Bindu to arrive at the same router again, it will take $N \times r \times p$ cycles, where 'N' is the number of nodes, 'r' is the router radix and 'p' is the Bindu movement period. During this time, if the network is not congested, a packet will definitely move forward two hops (even if N and r are small, p can be set to a large enough value to ensure two hops). If the end points are congested, it is theoretically possible, though extremely unlikely, for the same packet to be stuck at a router till the Bindu traverses the entire network and returns, and get misrouted again. However, as long as the network is deadlock-free (proven above), it cannot be congested indefinitely, thereby ensuring that eventually the packet will move forward two hops, and thus not livelock.

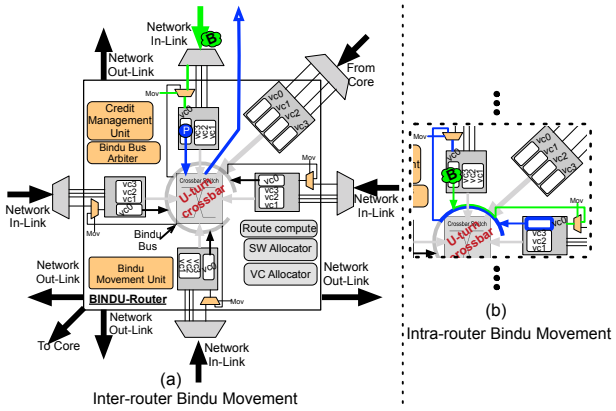

Figure 4: Router micro-architecture of BINDU. Additional components over baseline router are highlighted

3.6 Router micro-architecture

The router micro-architecture is shown in Fig. 4. We discuss the key modules incorporated to facilitate Bindu movement:

Bindu-bus: This is a bus connecting all the input ports, and is used to facilitate movement of the Bindu between VC-0 of the input ports (by moving the packet into the VC occupied by Bindu previously). A Bus suffices since only one Bindu can move per cycle; if multiple Bindus are concurrently present at a router, their period is skewed such that they do not contend for the bus in the same cycle.

Credit management unit: The upstream router is agnostic to the fact that there is a Bindu or an actual packet sitting at the downstream router. This implies that whenever Bindu replaces a packet in the router, there is no need to send updated credit signals to upstream routers involved. However, when Bindu replaces an empty slot, within the router, then both upstream routers, the one connected to the input port where Bindu was originally present and the other connected to the input port which originally had an empty slot, needs to be updated with credits accordingly.

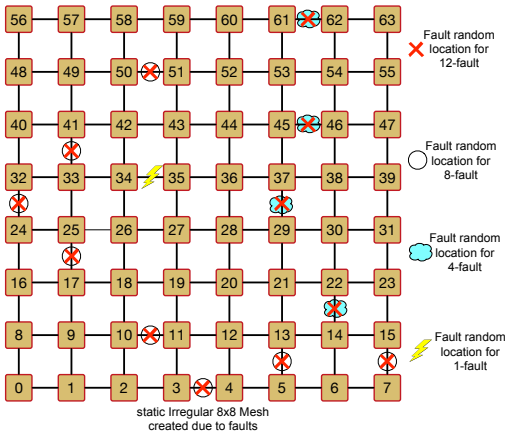

Figure 5: The figure shows irregular topologies, created out of a regular mesh. Faults in the network are shown as link failures at a random location, distributed randomly throughout the topology

Bindu movement unit: It is responsible for the overall movement of Bindu within the router until it leaves the router at a specific period. It comprises of two registers - one holding the Bindu movement period, and the other encoding its path within the router, including the output port connecting to the next router to route the Bindu to.

Multi-flit packets with Virtual Cut-Through (VCT) Routers. If the head-flit is present in VC-0 and it is the turn of this VC to turn into a Bindu, the VC is locked till the entire packet arrives and is not allowed to take part in switch arbitration. Once the entire packet arrives, transfer of this packet into the VC previously occupied by the Bindu is performed. The intra router Bindu period is chosen appropriately at design time such that entire packet can arrive and move before the next movement. If the head flit has already left, then the packet is allowed to naturally drain into its downstream VC without moving into the Bindu VC.

Multi-flit packets with Wormhole Routers: BINDU, as defined so far, works if VC-0 in each router is VCT, i.e., sized to hold complete packets (while other VCs can be smaller). To implement BINDU in wormhole routers, we would need to support packet truncation within VC-0, like prior works in deflection routing [13].

Bindu Movement Example. We explain the Bindu movement within and across the router, with the example shown in Fig. 4.

(1) A Bindu enters the router from the North input port - this effectively means that a packet sitting at the North input port leaves the router to go sit in the South input port of the upstream router, in place of the Bindu, as shown in Fig. 4(a).

(2) The Bubble Movement Unit encodes the period and the route of the Bindu within this router. In the example in Fig. 4(b), it moves the Bindu from the North to the East input port, via the Bindu bus. This step is similar to BBR [14].

(3) The Bindu will traverse all input ports sequentially. The last stop of the Bindu will be the input port, whose corresponding output port is connected to the next router in Bindu's path.

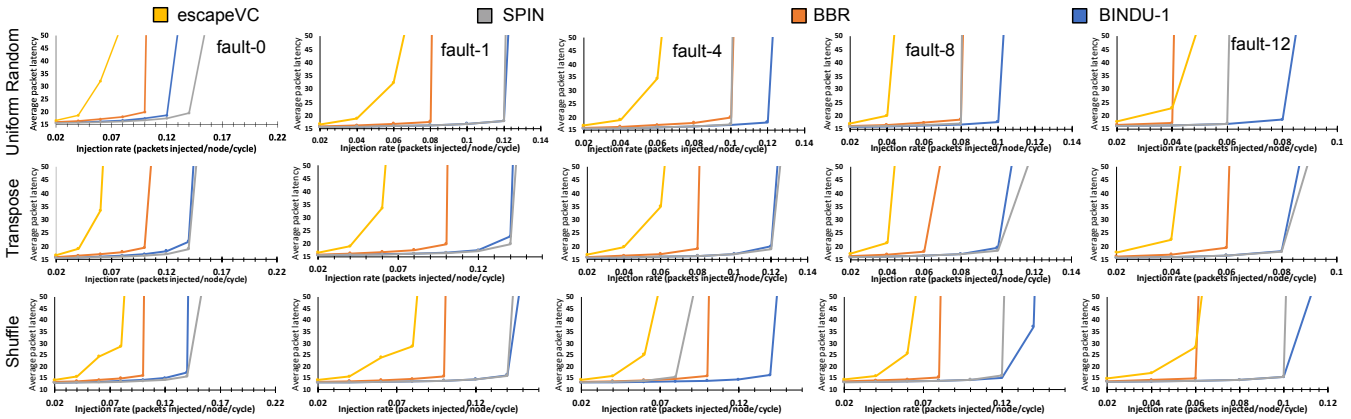

Figure 6: Performance of BINDU compared against Deadlock avoidance, Deadlock recovery and BBR for synthetic traffic: Uniform-Random, Transpose and Shuffle. Evaluated for vc=2, 64 node irregular topology derived from 8x8 Mesh.

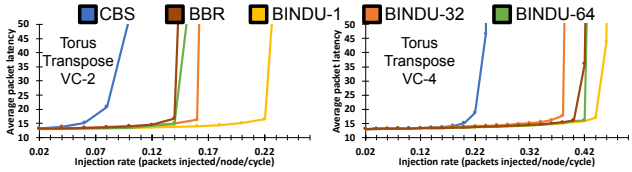

Figure 7: The graph compares the performance of BINDU with num Bindu=1, 32, 64 respectively with Critical Bubble Scheme and BBR. Graphs are for regular 8x8 Torus topology.

(4) The Bindu will use the crossbar to traverse to the neighbor by moving a packet (or an empty slot) at VC-0 of the corresponding input port from the neighbor to the current location occupied by Bindu, as shown in Fig. 4(a).
(5) This whole process is now repeated at the neighboring router.

Implementation Cost. The hardware overhead of BINDU comes because of addition of *Bindu-bus*, and *Bindu Movement Unit*. We used DSENT [21] to estimate the area and power overhead. Area overhead comes around 6% and static power overhead is 5% over baseline router with VC=4.

Implementation choice for Bindu: In this work, we proposed to embed Bindu-path inside the router, and Bindu moves through the network along that specified path. Another implementation choice could be to think of Bindu as a dummy packet which moves throughout the network in a cyclic manner and never gets consumed. We then could have the path for Bindu embedded inside the Bindu itself. This would further simplify the router micro-architecture of BINDU, as each router would then only need to read the content of Bindu to know where to route it next. It would also be easier to reconfigure Bindu's path dynamically by updating the route within Bindu based on some metric. However, the flexibility and reconfigurability comes at the cost of scalability; encoding the Bindu-path will need $logN \times logr \times r$-bits (where N is the number of nodes, and r is the router radix), which can exceed typical flit sizes for large networks.

Implementation of Bindu-Path: There could also be multiple ways in which Bindu path can be implemented as shown in Fig. 1. These paths would have different network-performance sensitivity for different irregular topologies. All these design choices are interesting to explore in future work, we, however, do not present these studies in this paper in the interest of space. We assume a snake-like structure for regular topologies, and a tree for irregular topologies.

Table 2: Qualitative Comparisons of CBS, BBR and BINDU

	CBS [7]	**BBR** [14]	**BINDU**
Bubble Implementation	Bubble is an empty VC	Bubble is empty VC	Bindu is reserved VC or dummy packet
Movement	Bubble moves naturally as the packet moves	Random proactive bubble movement within router	Proactive Bindu movement as per Bindu-path.
Minimum Empty Buffers	one bubble per ring dimension. 8x8 torus network requires 32 bubbles	one bubble per router. 8x8 torus requires 64 bubbles	one Bindu in the entire network. 8x8 torus requires one Bindu
Topologies	closed loop/ring topologies, for example Torus	Works for any arbitrary topology	Works for any arbitrary topology
Routing restriction	uses dimensional order routing in the VC that contains the bubble	uses minimal random adaptive routing	BINDU uses minimal random adaptive routing
Misrouting	No packet gets mis-routed	in-frequent misroute during bubble exchange	At most one packet per Bindu movement
Flexibility	not flexible	not flexible	flexible in terms of number of Bindus and their path
Reconfigurable	not reconfigurable	not reconfigurable	dynamically reconfigurable

3.7 Comparison with CBS and BBR

Here we delineate BINDU from two notable works which use bubble to provide deadlock-freedom to the network. We paraphrase the main points in Table 2. BBR can be viewed as Bindu-64.

4 EVALUATIONS

4.1 Methodology

BINDU is evaluated using *gem5* [5] with the *Garnet2.0* [1] network model and the *Ruby* memory model. We use *DSENT* [21] to model power and area for a 11 nm process. Table 3 lists all key configuration parameters for our evaluation.

Baseline Networks. We select state-of-the-art baseline deadlock-free networks to compare against BINDU. From deadlock-avoidance, we use escape VCs (which are known to perform better than turn-restriction schemes [16, 17]). From deadlock-resolution, we choose SPIN [17], which has been shown to performs better than Static Bubble [16]. From bubble-based, we choose BBR [14] and CBS [7]. BBR works for arbitrary topologies while CBS only for Torii.

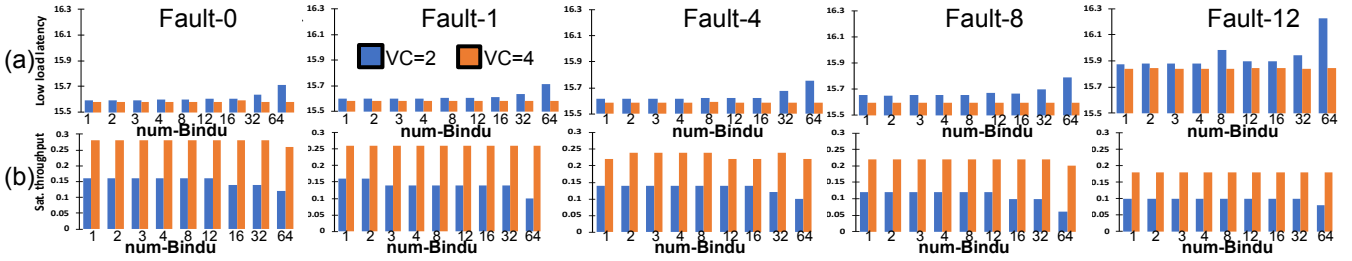

Figure 8: Graphs are for Transpose traffic pattern as number of Bindus increase from 1 to 64 in 8x8 irregular Mesh topologies with given fault. Graph-(a) shows the effect of low-load latency. We observe that the affect of number of bubbles on performance, is more for the router with fewer VCs compared to the router with more VCs per input port. All Bindus in BINDU are confined to VC-0 of each input port. Graph-(b) shows the effect of saturation throughput. We observe that with increase in number of Bindus, saturation throughput decreases. Bindu-64 is similar to BBR

Table 3: Key Simulation Parameters.

Real application simulation parameters	
Core	64 cores and RISCV ISA (Ligra), 1GHz 16 cores and x86 ISA (Parsec3.0), 1GHz
L1 Cache	Private, 16KB Ins. + 16KB Data, 4-way set assoc.
Last Level Cache	Shared, distributed, 64KB, 8-way set assoc.
Cache Block Size	64B
Cache Coherence	MESI Directory (Ligra) Vnets=5 MOESI hammer (Parsec3.0) Vnets=6
Target Networks	
Topology	irregular 8x8 Mesh (Ligra and Synthetic workloads) irregular 4x4 Mesh (Parsec3.0)
Router latency	1-cycle
Num VCs	1, 2 and 4
Buffer Organization	Virtual Cut Through. Single packet per VC
Link Bandwidth	128 bits/cycle
Deadlock Avoidance **Deadlock Recovery** **Bubble based**	Escape VC [11] with Up-Down [18] within Esc-VC SPIN [17] CBS [7]; BBR [14]; BINDU-k (k: num of Bindus)

Benchmarks. Both real applications and synthetic traffic are used to evaluate BINDU. Applications are drawn from the Ligra benchmark suites [20] and from Parsec3.0 [4] . For synthetic traffic, we focus on *uniform random*, *transpose* and *shuffle* traffic with the mix of 1-flit and 5-flit packet size; results for other traffic patterns are qualitatively similar. The simulator is warmed-up for 1000 cycles, thereafter network statistics are collected by injected fixed number of tagged packets by each node in the system. Simulation completes when all the tagged packets are received. We use an 8×8 irregular network for Ligra and synthetic traffic. Ligra applications have been simulated in syscall-emulation (SE) mode of gem5 while Parsec3.0 applications are simulated on irregular 4x4 network using full system simulation mode in gem5.

Topologies. BINDU performance is evaluated on fault-free and faulty 2D mesh networks as well as 2D Torus network topology.To create irregular topologies from 2D mesh, faults are injected randomly into the network while network connectivity is maintained as shown in Fig. 5. For the 8×8 network, we consider a range of faulty links up to 12 in 2D mesh.

4.2 Performance

Irregular topologies. Fig. 6 shows the performance comparison of BINDU with other state of the art deadlock-freedom schemes. for irregular 2D 8x8 Mesh. Key-takeaway from this performance graph is that except regular 2D Mesh, BINDU performs comparably to the state of the art solutions. In fact, under certain traffic pattern for example uniform random and shuffle, BINDU performs better than state-of-the-art. On average, we see 15% improvement over saturation throughput in synthetic traffic pattern using BINDU.

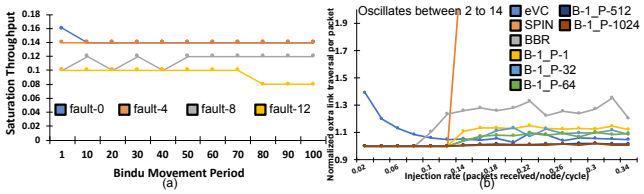

Figure 9: (a)Sensitivity of saturation throughput with increase in inter-router Bindu movement period of one Bindu for uniform random traffic. These results are for irregular 8x8 Mesh with VC=2. (b)Uniform-random traffic, VC=2, with Fault-1. The graph shows the extra link traversal over the baseline using minimal deadlock-free routing. Here *B-1_P-X* means Bindu-1 with 'X' as Bindu Movement Period

Bubble-based Schemes. Fig. 7 compares the performance of state-of-the-art bubble based deadlock-freedom techniques with BINDU for regular 8x8 Torus topology. Since CBS has 32, and BBR has 64 bubbles, we also contrast against iso-Bindu configurations. BINDU provides up to 2.2× higher throughput. Also, the performance of BINDU decreases as the number of Bindus increases in the topology. BBR can be approximately considered as Bindu-64 as now each router has a Bindu/bubble. Therefore, BBR's performance can be approximated with Bindu-64, and Bindu-32 lies between Bindu-1 and BBR. We observe 35% throughput improvement for VC=2 and 15% higher throughput for VC=4 with BINDU-1 compared to BBR.

4.3 Sensitivity studies

4.3.1 Number of bubbles. Fig. 8(a) shows the sweep of low load latency as the number of Bindus increases in the topology from one to 64. In general, we see low load latency increases with an increase in the number of Bindus, both for regular as well as irregular 8x8 Mesh topology. Affect on low load latency is more prominent for VC=2 than VC=4. Also, sensitivity reduces for higher fragmented topology (for example fault-12) compared to lower fragmented topology (for example fault-1). Similar trends are shown by saturation throughput as the number of Bindus increases, saturation throughput decreases, for both regular and irregular Mesh topology in Fig. 8(b). This happens mainly because of two reasons. Firstly, as the number of Bindus increases, more packets will be misrouted in the network. Secondly, Bindus cannot be consumed therefore they put indirect restrictions on the number of packets that can be injected into the network. Since Bindu only stays in VC-0, we see less sensitivity in saturation throughput when there are many VCs (for example VC=4) compared to when there are fewer VCs (for example VC=2).

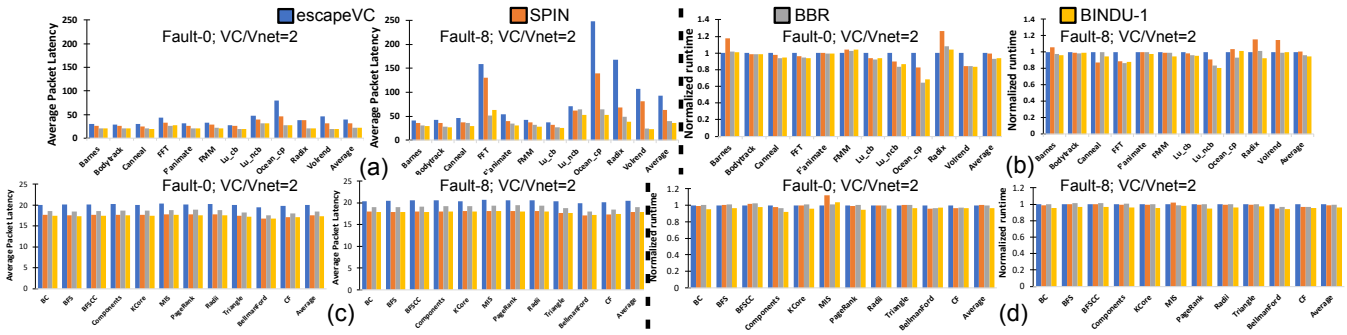

Figure 10: Packet latency from real workloads and their normalized runtime improvement with BINDU when compared to other state of the art schemes. (a) and (b) are the packet latency and normalized runtime for Parsec while (c) and (d) are the same metrics for Ligra

4.3.2 Inter-router Bindu movement period sweep. Fig. 9-(a) shows the sensitivity of saturation throughput with increase in inter-router Bindu period. The experiment is performed for uniform random traffic pattern with VC=2, with one Bindu in the whole network. We observe decrease in saturation throughput with increase in inter-router Bindu-period. This happens because it takes longer for the Bindu to reach to the packets stuck in deadlock and free them from deadlock.

4.3.3 Energy Overhead. Fig. 9-(b) shows the energy overhead for various schemes, in the form of extra link-traversal per packet, over the baseline assuming *ideal minimal routing* without any overhead for an irregular mesh with one fault. For escape VC the extra link traversal comes because of the non-minimal Up-Down [18] path a packet takes to reach its destination within the Escape VC. We observe higher overhead at lower injection rate because the fewer the packets in the network, the more sensitive they are to non-minimal path of escape VC. SPIN's [17] overhead over ideal minimal routing is because of probes used for detecting deadlock, especially at higher loads due to increased forking at intermediate routers. BBR's [14] link traversal overhead is because of increased *bubble-exchange* at high injection rate. BINDU's extra link traversal over ideal minimal routing is because at higher injection rate, there are more packets in the network, hence the likelihood of Bindu replacing a packet, as it moves along its path, increases. In summary, we can see that the additional energy expended by the misroutes due to Bindu movement is negligible at low-loads, and less than 10% post saturation (even with an aggressive Bindu movement period of 1), which is either equal to or much lower than other state-of-the-art solutions.

4.4 Real application results

Fig. 10-(a) and (b) show the packet latency and normalized runtime improvement for Parsec3.0 benchmarks respectively. This culminates into 6% average improvement in runtime for fault-0 and 7% average improvement in the runtime for fault-8 in 4x4 mesh respectively. Fig. 10-(c) and (d) show the packet latency and normalized runtime improvement for Ligra applications respectively. There is overall around 5% improvement of BINDU over other schemes, in both average total packet latency and runtime of the application.

5 CONCLUSION

BINDU is the first work, to the best of our knowledge, to demonstrate deadlock freedom by reserving a single bubble (empty VC) in the entire network, and pro-actively moving it through all routers and all input ports. BINDU requires no deadlock-detection (unlike deadlock recovery schemes) and requires no turn-restrictions or escape VCs (unlike deadlock avoidance schemes). BINDU is topology-agnostic and provides around 15% average throughput improvement over state-of-the-art techniques in synthetic traffic, and around 7% improvement on an average runtime of real applications. This makes BINDU an effective solution to implement in irregular network topologies to guarantee deadlock-freedom.

REFERENCES

[1] N. Agarwal *et al.* 2009. GARNET: A Detailed On-chip Network Model inside a Full-system Simulator. In *ISPASS*.

[2] R. Al-Dujaily *et al.* 2012. Embedded Transitive Closure Network for Runtime Deadlock Detection in Networks-on-Chip. *IEEE Transactions on Parallel and Distributed Systems*.

[3] K. V. Anjan and Timothy Mark Pinkston. 1995. An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA. In *ISCA*.

[4] C. Bienia *et al.* 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT*.

[5] N. Binkert *et al.* 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011).

[6] Lizhong Chen and Timothy M. Pinkston. 2013. Worm-Bubble Flow Control. In *HPCA*. 366–377.

[7] L. Chen *et al.* . 2011. Critical Bubble Scheme: An Efficient Implementation of Globally Aware Network Flow Control. In *IPDPS*. 592–603.

[8] W. J. Dally and H. Aoki. 1993. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE TPDS* 4, 4 (April 1993), 466–475.

[9] W. J. Dally and C. L. Seitz. 1987. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Comput.* (1987), 547–553.

[10] Jose Duato. 1993. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. Parallel Distrib. Syst.* (1993).

[11] Jose Duato. 1995. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. Parallel Distrib. Syst.* 6, 10 (Oct. 1995), 1055–1067.

[12] Christopher J. Glass and Lionel M. Ni. 1994. The Turn Model for Adaptive Routing. *J. ACM* 41, 5 (Sept. 1994).

[13] Thomas Moscibroda and Onur Mutlu. 2009. A Case for Bufferless Routing in On-chip Networks. In *ISCA*.

[14] M. Parasar, A. Sinha, and T. Krishna. 2018. Brownian Bubble Router: Enabling Deadlock Freedom via Guaranteed Forward Progress. In *NOCS*. 1–8.

[15] V. Puente *et al.* 2001. The Adaptive Bubble Router. *J. Parallel Distrib. Comput.* 61, 9 (Sept. 2001).

[16] Aniruddh Ramrakhyani and Tushar Krishna. 2017. Static Bubble: A Framework for Deadlock-Free Irregular On-chip Topologies. In *HPCA*. 253–264.

[17] A. Ramrakhyani *et al.* 2018. Synchronized Progress in Interconnection Networks (SPIN) : A New Theory for Deadlock Freedom. In *ISCA*.

[18] M. D. Schroeder *et al.* 1991. Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links. *J-SAC* 9, 8 (1991).

[19] D. Seo *et al.* 2005. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *ISCA*.

[20] Julian Shun and Guy E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. *SIGPLAN Not.* 48, 8 (2013), 135–146.

[21] C. Sun *et al.* 2012. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *NOCS*.

[22] R. Wang *et al.* 2013. Bubble Coloring: Avoiding Routing- and Protocol-induced Deadlocks with Minimal Virtual Channel Requirement. In *ICS '13*.

[23] Yuankun Xue and Paul Bogdan. 2015. User cooperation network coding approach for NoC performance improvement. In *NOCS*. ACM, 17.