SWAP: Synchronized Weaving of Adjacent Packets for Network Deadlock Resolution

Mayank Parasar[1], **Natalie Enright Jerger**[2], **Paul V. Gratz**[3], **Joshua San Miguel**[4], and **Tushar Krishna**[1]

[1]School of ECE at Georgia Institute of Technology, [2]Department of ECE at the University of Toronto,

[3]Department of ECE at Texas A&M University, [4]Department of ECE at the University of Wisconsin-Madison

[1]*mparasar3@gatech.edu* [2]*enright@ece.utoronto.ca* [3]*pgratz@tamu.edu* [4]*jsanmiguel@wisc.edu* [1]*tushar@ece.gatech.edu*

ABSTRACT

An interconnection network forms the communication backbone in both on-chip and off-chip systems. In networks, congestion causes packets to be blocked. Indefinite blocking can occur if cyclic dependencies exist, leading to deadlock. All modern networks devote resources to either avoid deadlock by eliminating cyclic dependences or to detect and recover from it.

Conventional buffered flow control does not allow a blocked packet to move forward unless the buffer at the next hop is guaranteed to be free. We introduce SWAP, a novel mechanism for enabling a blocked packet to perform an in-place swap with a buffered packet at the next hop. We prove that in-place swaps are sufficient to break any deadlock and are agnostic to the underlying topology or routing algorithm. This makes SWAP applicable across homogeneous or heterogeneous on-chip and off-chip topologies. We present a lightweight implementation of SWAP that reuses conventional router resources with minor additions to enable these swaps. The additional path diversity provided by SWAP provides 20-80% higher throughput with synthetic traffic patterns across regular and irregular topologies compared to baseline escape VC based solutions, and consumes 2-8× lower network energy compared to deflection and global-synchronization based solutions.

1. INTRODUCTION

Interconnection networks form the communication backbone of today's computer systems. They provide the means for communication between different entities of the system, be they different cores of a many-core chip multiprocessor (CMP), or a system-wide network where each node itself could be a CMP or server.

In a buffered network, packets naturally get blocked due to contention for shared links. A *deadlock* occurs when some packets remain blocked inside the network indefinitely due to a cyclic dependence between buffers, and never reach their destination, causing application or system level failure. Deadlock freedom is a matter of *correctness*, rather than performance. Therefore, careful consideration is given while designing the network to make it deadlock free; more often than not, via over provisioning of resources.

Almost every commercial interconnection network today *prevents* deadlocks from occurring by ensuring that the aforementioned cyclic dependence between buffers never gets created in the first place. This is done by restricting certain turns (either for all packets such as with dimension-ordered XY routing [1], or within extra *escape* virtual channels (VCs) [2]) or by restricting the injection of packets [3,4]. These restrictions reduce path diversity; throughput is lost since avoiding runtime formation of cycles means that links will go unused by certain packets even if they are idle. Moreover, they are inherently tied to the topology (e.g., XY routing only works for a mesh; any other topology requires a full channel dependence graph analysis to disable turns [1], BFC only works in rings/tori) – making them inflexible as plug-and-play solutions in *arbitrary* topologies in heterogeneous SoCs or when links/routers fail due to waning silicon reliability [5].

We focus on schemes that provide full path diversity, and *resolve* deadlocks that have occurred. We characterize prior work on deadlock resolution via the following taxonomy:[1]

- Deadlock resolution via **escaping** (e.g., escape buffers [5, 6]); the key drawback is the need for extra buffers;
- Deadlock resolution via **misrouting** (e.g., deflection routing [7,8,10]); the key drawback is increased energy consumption and loss in throughput;
- Deadlock resolution via **coordinating or synchronizing** (e.g., SPIN [9]); the key drawback is expensive global coordination for detection and spinning.

None of the state-of-the-art deadlock-freedom solutions (avoidance/recovery) provide full path diversity and high-throughput without requiring deadlock detection or global coordination for arbitrary topologies. This motivates our work. Going back to first principles, we argue that a network deadlocks because a packet is indefinitely blocked; the reason is a fundamental network design rule that says that a packet should be forwarded from an upstream router to a downstream router *if and only if* the downstream buffer is free (which is known via credits/on-off signaling), or is guaranteed to become free by the time the packet arrives [9,11]. If this rule is violated, packets may be dropped.

We question this fundamental rule and propose *Synchro-*

[1]Section 2 provides more details on each of these approaches.

MICRO '52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10...$15.00
https://doi.org/10.1145/3352460.3358255

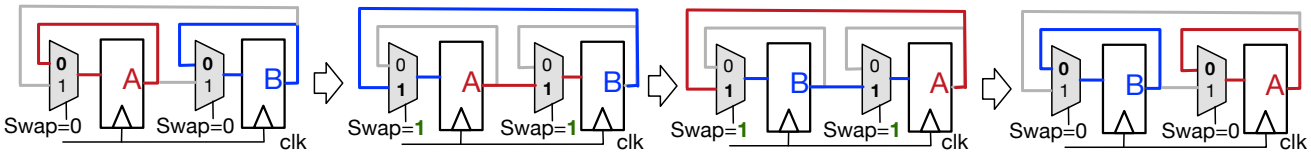

Figure 1: **The basic hardware implementation for swapping the content of two Flip Flops / FIFOs.**

Table 1: **Qualitative Comparison of Deadlock Freedom Mechanisms**

Deadlock Freedom Mechanism	Resolution Type	Full Path Diversity	No VCs Required	Topology Agnostic	No coordination
CDG / Dally [1]	No Cycle	✗	✓	✗	✓
Duato [2, 6]	Escape	✓/✗*	✗	✓/✗*	✓
BFC [3, 4, 5]	Bubble	✗	✓	✗	✓
Deflection [7, 8]	Misroute	✗**	✓	✓	✓
SPIN [9]	Synchronize	✓	✓	✓	✗
SWAP	Backtrack	✓	✓	✓	✓

* Within esc VC: limited path diversity + requires topology info for esc path.

**At low-loads, full path diversity is available. But at medium-high loads, packets cannot control the directions or paths along with they are deflected.

nized Weaving of Adjacent Packets (SWAP). SWAP introduces a new deadlock resolution paradigm: **backtracking**, where deadlocked packets yield their position and allow other packets to move forward. Backtracking is performed via *in-place packet swaps* across buffers in neighboring routers. Fig. 1 shows the conceptual idea. Unlike software, where a swap requires additional temporary storage, in hardware, an in-place swap is conceptually the same as a cyclic shift register. Intuitively, SWAP allows any blocked packet to make *guaranteed* forward progress to its destination, regardless of the congestion in the network, via the mechanism of swaps. More formally, we prove that performing swaps at periodic intervals ensures that any cycles, if they form in the network, are broken dynamically via swaps, guaranteeing deadlock freedom. Further we show that despite infrequent misrouting of packets, SWAP is free from livelock.

This paper makes the following contributions:

- We propose SWAP, a novel mechanism for in-place packet swaps across routers in a network.
- SWAP provides deadlock-freedom via packet backtracking, while providing desired metrics of full path diversity, high throughput, no additional VCs, no deadlock detection, and topology agnosticism. Table 1 qualitatively contrasts SWAP with current deadlock-freedom solutions.
- We present a light-weight implementation of SWAP that adds 4% area overhead over a state-of-the-art VC router.
- SWAP increases throughput by 20-80% with synthetic benchmarks for a full and faulty mesh, compared to escape VCs, and reduces network link activity by 2-8× compared to deflection and synchronization based schemes.
- We show that using SWAP with conventional deadlock-free routing enhances network throughput by 10% since swaps allow packets to move away from congested parts of the network. Thus SWAP *emulates the behavior of VCs* without adding any additional buffers, making it applicable beyond deadlock resolution.

2. BACKGROUND AND RELATED WORK

There exists a significant body of work on deadlock freedom in interconnection networks. For the purpose of discussion, we categorize them into *deadlock avoidance* and *deadlock resolution* techniques as per our taxonomy in Section 1. Table 1 summarizes key attributes of different approaches.

2.1 Deadlock Avoidance

Routing Restrictions. The most common technique to avoid deadlocks is to make the *Channel Dependency Graph* (CDG) acyclic [1]. In one variant of this technique, certain *turns* in a given topology are not allowed, to ensure that a deadlock is never created. The *turn model* [12] for a mesh is the most prevalent implementation. These algorithms allow *selective adaptivity* in the routing algorithm–for example, west-first routing only allows adaptivity if the destination happens to be in the North-East or South-East quadrant of the mesh. An alternate implementation is to change the *virtual channel (VC)* at which the packet would sit at downstream router whenever certain turns are made, to ensure that the VCs themselves do not form a cyclic dependence. This is used in off-chip networks for algorithms such as UGAL [13] in dragon-fly networks and require at least three VCs. *Fully adaptive* routing can be implemented to allow full path diversity across all VCs, while each VC itself has turn restrictions [14]. In irregular topologies, arising due to network faults [15, 16, 17] or power-gated nodes [18], spanning trees are often used to guarantee the same acyclic CDG behavior.

Injection Restrictions. Another technique to avoid deadlocks is to ensure that a cyclic dependence never forms at runtime even though the CDG is cyclic. This can be ensured by cleverly managing the injection of packets into the network. Bubble flow control [3] and its variants [3, 4, 19, 20] are the most common implementation of this idea, but only work for rings and tori. Some extensions of BFC for meshes [21] and dragon-fly [22] have been explored.

Both of these approaches limit throughput either by restricting path diversity or by limiting packet injection. They also tend to be topology-specific, limiting their use in irregular or faulty networks.

2.2 Deadlock Resolution

Escape Virtual Channels. Escape VCs [2, 23, 24] allow all VCs to use deadlock-prone routing with no turn restrictions, except one (the escape VC) that uses a deadlock-free routing path. This provides an acyclic CDG only within the escape VC. Thus, escape VC-based solutions require at least 2 VCs to provide *fully adaptive* routing. Escape VCs have been used across a suite of networks [24, 25, 26, 27, 28, 29, 30]. A key challenge with escape VCs, just like the CDG schemes, is that they are topology-dependent–the path through the

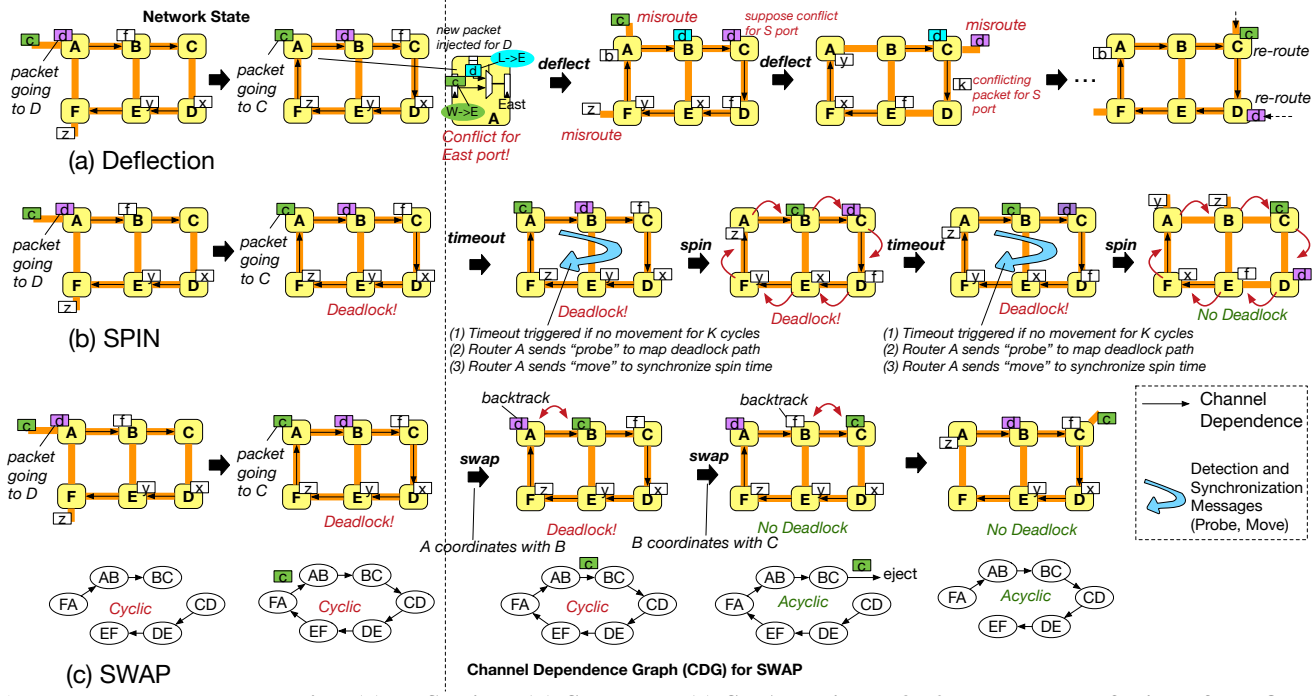

Figure 2: **Example comparing (a) Deflection, (b) SPIN and (c) SWAP using a 3×2 mesh. The left side of the figure (before dotted line) sets up the same initial condition of deadlock in the three designs, and the right side demonstrates how they operate. In deflection routing, the deadlock does not persist as packets move every cycle. However, the green packet (going to Router C) and the purple packet (going to Router D) are both misrouted due to conflicts, and take multiple cycles to be re-routed to their destinations. In SPIN, if a packet in a specific VC (e.g., at Router A) does not move for a specified number of cycles, a timeout occurs, and a probe is sent to map the possible deadlock path. The probe returns after 12 cycles. A move message synchronizes all routers on the deadlock path to perform a spin. Once the move returns, the spin is performed, and every packet moves forward one hop. The deadlock still persists, so the timeout, probe, move, and spin process repeats. In the last step, packet c reaches its destination and the deadlock is resolved. In SWAP, Router A (at a fixed period), coordinates locally with its neighbor (Router B) and performs a swap: packet d is backtracked and packet c moves forward. The deadlock still persists. Packet c performs another swap, reaches its destination, and the deadlock is resolved. The corresponding CDG at every step in SWAP is also shown.**

escape VCs needs to be deadlock-free via routing restrictions. Further, they require the overhead of provisioning at least two VCs, one of which typically is not well utilized.

Misrouting packets (Deflection routing). Deflection routing [7,8,31,32] assigns every input flit 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. It is primarily intended for bufferless routers. It is inherently deadlock free because every packet makes progress every cycle. Although it is not guaranteed that same packet will make forward progress at each router. This increases network congestion and energy consumption due to mis-routing which will be especially bad at high loads, limiting network throughput significantly [33].

Coordination. Deadlock detection and recovery is an alternative approach to provide deadlock freedom. These are motivated by the fact that deadlocks are actually quite rare [5,6] and argue that the routing restrictions or additional escape VCs are overly conservative. These solutions fundamentally rely on a deadlock detection mechanism [5,9,34,35] involving time-outs and probes, followed by a recovery mechanisms that introduces additional buffers [5,6,35] or coordinates packet movement [9] to guarantee forward progress. Although their datapath overheads are low, these solutions require extremely complex control circuitry in the form of time-out counters and state machines to detect deadlocks and manage false positives and negatives. Moreover, due to the overhead of deadlock detection, they suffer from low throughput once the network starts to deadlock. Thus, they have not made it into mainstream systems.

Backtracking. SWAP uses the same underlying theory of guaranteed forward progress as deflection routing to provide deadlock freedom–thereby not requiring any turn or injection restrictions, or detection and recovery. Packet swaps can be viewed at a high level as *controlled* deflections. There is a subtle yet important difference: misrouting is tricky because you have to make sure that you misroute a packet only to directions where it still has a legal path to its final destination; whereas backtracking guarantees this. The key differences of SWAP compared to a deflection-based mechanism and SPIN [9] (coordination-based) are highlighted in Fig. 2 and discussed in detail in Section 4.4.

3. SWAP THEORY

In this section, we present the theoretical underpinnings of our SWAP scheme. First, we provide necessary definitions for the reader, overview the basic operation and provide a

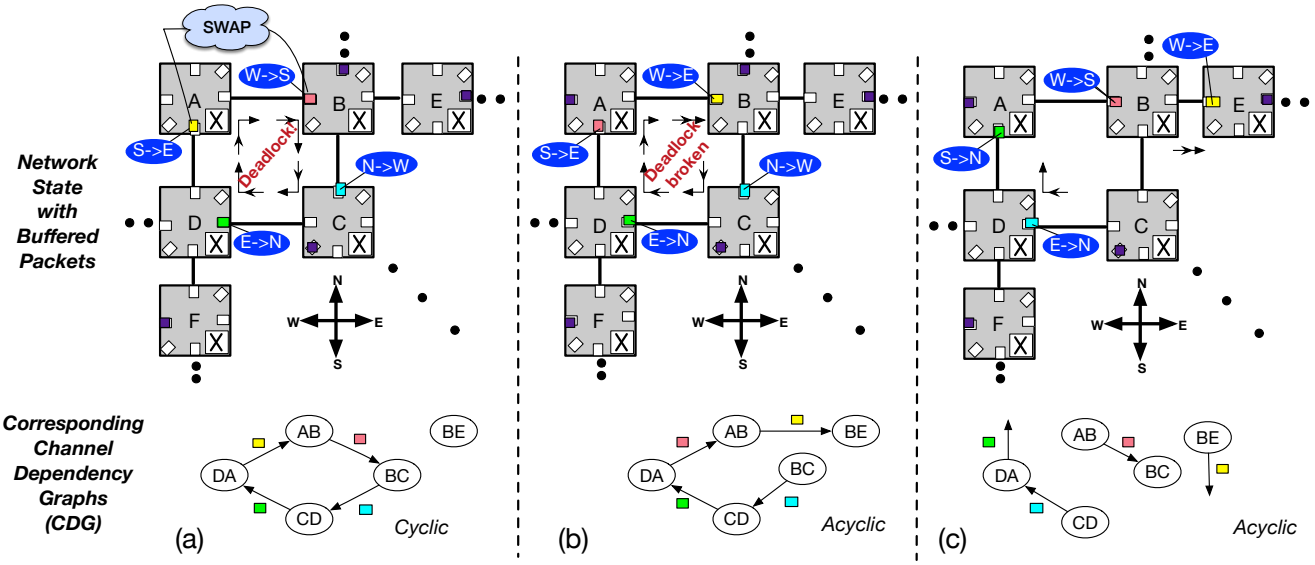

Figure 3: **Walk through example of SWAP with corresponding CDG. Each node in the CDG represents a link (e.g., node '*AB*' is the link from *router-A* to *router-B*) and each edge represents a packet that wants to turn from the source link to target link (e.g., '*AB*' to '*BC*' represents the pink packet currently buffered at router-B making a West to South turn). (a) there is a deadlock between the four packets as seen by the cyclic CDG. A swap is initiated by router-A between the yellow packet at A with the pink packet at B. (b) The swap completes. Now the yellow packet (swapFwd) moves to B and wants to go East, while the pink packet (swapBack) is *backtracked* to A. The CDG is acyclic: the deadlock is broken. (c) All packets move forward via normal operation.**

concrete walk-through example. Then we provide a proof of deadlock and livelock freedom. Section 4 then presents one possible realization of our SWAP theory.

3.1 Definitions

Deadlock: Deadlock occurs when packets remain inside the network indefinitely and never reach their destination. Packets wait forever to acquire the buffer at next router due to a *cyclic dependency* between the buffers.

Forward Progress: We use forward progress for a packet to refer to a scenario where it moves towards its destination.

Backtrack: We refer to backtracking for the packet that yields its position and moves back to the upstream router during the swap.

Swap: A swap refers to the act of interchanging two packets from two adjacent routers. The high-level idea is shown in Fig. 1. *A Swap requires no additional buffers*. It leverages the bi-directional links between adjacent routers to *simultaneously* send two packets (serializing the flits over the link) in either direction.

swapFwd packet: During a swap, the initiator (a.k.a. upstream router) chooses and routes the swapFwd packet towards the productive direction to the downstream router. This packet makes *forward progress.*

swapBack packet: The packet backtracked from the downstream router to the initiator to allow *swapFwd packet* to sit in its place. The *swapBack packet* acquires the buffer of upstream router which was held by the *swapFwd packet*.

swapCycle: The cycle during which a specific upstream router performs a swap of one of its buffered packets.

swapPeriod: *swapPeriod* refers to the number of cycles it takes for the same router in the network to try and initiate a swap for one of its buffered packets.

3.2 Proof of Deadlock Freedom

Theorem 1: *In a deadlock cycle of length n, at most $n-1$ swaps by a specific packet are sufficient to break the deadlock.*

Proof: A swap operation removes one edge from and adds one new edge to the runtime CDG. The edge that is removed is the original direction the swapBack packet intended to go towards, while the edge that is added is the new direction the swapFwd packet intends to go towards. $n-1$ swaps allows the packet to reach the node one-hop behind it. One of these $n-1$ intermediate routers will either be its final destination, or an exit point out of this ring. In either scenario, the swapFwd packet will no longer have an edge in the CDG along the original dependence cycle, thereby breaking the deadlock.

Example: Fig. 3 shows the network state along with the channel dependence graph (CDG) at each step. For the sake of simplicity, we show one VC per port, and single-flit packets. In Step (a), four packets are in a deadlock, as is also evident by the cyclic CDG. Suppose Router A chooses the yellow buffered packet as its swapFwd packet. The yellow packet wishes to go East making the pink packet at the downstream router the swapBack packet. A sends a swap request to B, receives an ACK, and the swap is executed. In Step (b), the yellow packet has made forward progress; the deadlock is broken as it wants to go East. This is also seen from the CDG. The packets now move forward via conventional means, as seen in Step (c). Here, the first swap led to the CDG becoming acyclic, since the new edge was no longer along the original cycle. In a more general case, it may be possible that even after a swap, the CDG remains cyclic (for example, if there is a longer dependence cycle, as shown in Fig. 2(c)).

Theorem 2: *For a given system which implements SWAP, as long as every packet gets a chance to perform a swap, the*

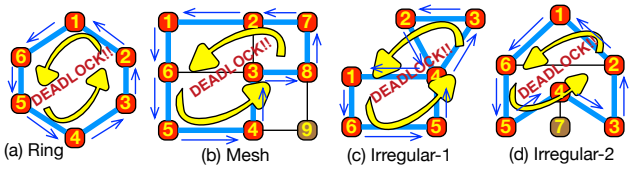

Figure 4: **Examples of Deadlocks in Arbitrary Topologies**

network is deadlock-free.

Proof: A deadlock, by definition, is an *indefinite* blocking of a packet. As long as the implementation of SWAP can guarantee that every packet will have a chance to perform a swap, it will make forward progress, making the network inherently deadlock free. However, it is possible for the packet that made forward progress to later be backtracked by another packet. Indefinite backtracking could lead to a livelock (i.e., the packet never reaches its destination). Next, we discuss why SWAP is livelock free.

3.3 Proof of Livelock Freedom

Theorem: *For a given system that implements SWAP, as long as any backtracked packet eventually has the opportunity to move forward by two hops before being backtracked again, the network is livelock-free.*

Proof: Backtracking can be viewed as incrementing a packet's number of remaining hops h (to its destination) by at most 1. Assume that the system allows a packet to move two hops before being backtracked again (i.e., before being selected as a swapBack packet). For every increment by 1 in h due to backtracking, there is a decrement by 2. This implies that in this system, for any given packet, h eventually goes to 0, guaranteeing forward progress to the destination.

Implementation: There are two important design considerations to implement such a system. First, the swapPeriod must be greater than the time it would take for a packet to move two hops forward in the absence of contention. This avoids any pathological cases of packets continuously being backtracked. Second, the selection scheme that decides which packet should be swapped next must be fair. Ideally, this selection is random; though for practical purposes, round robin is sufficient. A fair selection scheme ensures that no packet is starved in the presence of contention. Even if a packet is not able to move two hops forward now, it will eventually be able to, as guaranteed by the fair selection. Since the network topology is finite, h is upper-bounded by the network diameter (i.e., the largest number of hops between any two routers). Thus no amount of contention in the system can cause a packet to be backtracked indefinitely.

3.4 SWAP in Arbitrary Topologies

Arbitrary topologies are challenging for popular deadlock-avoidance solutions such as XY/West-first routing algorithms; routing algorithms now require CDG analysis to determine topology-specific turn restrictions (for all paths or within escape VCs). In contrast, SWAP is agnostic to the topology as any swap just involves neighboring routers. For example, the deadlock ring in Fig. 2 could lie within any topology in Fig. 4 and use the same mechanism of swaps for deadlock freedom. SWAP is also agnostic to the underlying routing algorithm. The routing algorithm decides the output port (i.e., neighboring router) of the swapFwd packet, and the swapBack packet is chosen from the corresponding input port at the neighbor.

4. SWAP IMPLEMENTATION

Multiple implementations of SWAP are possible. We favor an implementation with low complexity. We describe the possible design space and the intuition behind our given design choices, acknowledging that alternate implementations are possible.

4.1 Initiating a Swap

Although it is possible to map out the full deadlock loop at runtime via timeout and probes [5, 9], and then perform controlled swaps to recover from the deadlock (Proof 1 in Section 3.2), we prefer a less expensive approach. Recall that any SWAP implementation ensures deadlock and livelock freedom if it ensures that (a) every packet gets the chance to make forward progress via a swap, and (b) the system allows a packet to move two hops in an uncongested scenario, before being backtracked. To ensure (a), we enforce periodic swaps by every router at a configurable time period (swapPeriod) and we add a pointer in every router to cycle through all VCs at all ports that decides which VC will try and initiate a swap. To ensure (b), we need to account for the worst case delay for a packet in two adjacent routers without any stalls due to insufficient credits. This would be a packet in a VC contending with all other VCs at that router for a specific output port, followed by traversing the router and link, and repeating the same at the next router. Thus,

$$swapPeriod \geq 2 \times (\#\,ports \times \#\,vcs/port + (router\_pipeline\_delay + link\_delay)) + serialization\_delay$$

This works out to be 54 cycles for a 5-ported mesh router with 4 VCs per port, 5-flit packets, 4-cycle routers and 1-cycle links, and 18 cycles for a 1-cycle, 1 VC per port mesh router.

In our implementation, each router performs a swap during its **swapCycle**. The swapCycle is defined as

$$(cycle/m)\%(K \times N) == router\_id \quad (1)$$

where K is a configurable *swapDutyCycle*, N is the number of routers in the network, m is the maximum number of flits of any packet in the system and $K \times N$ is the *swapPeriod*. K determines how often each router initiates swaps; the lower the value of K (minimum could be 1), the more *swaps* performed in the network. When $K = 1$ and $m = 1$, each router initiates a swap every N cycles in a TDM manner. In a 64-core system, this means that even with $K = 1$, each router attempts a swap once every 64 cycles, which is greater than the minimum swapPeriod calculated above for livelock avoidance.

The router initiating the swap, as dictated by its ID and current cycle, is the *upstream router* and router with which it will swap its packet, is the *downstream router*. At any given cycle, by design, there can only be one upstream router and several possible downstream routers depending on the topology.

During the swapCycle, the upstream router selects a swapFwd packet from one of its internally buffered packets. It sends a swap request signal via a 1-bit wire to the downstream

router at the output port for this packet (which is determined by the routing algorithm). The downstream router selects a swapBack packet and sends an ACK. Section 4.2 details how the swapFwd and swapBack packets are selected. Upon receiving the ACK, a swap is executed over the next *m* cycles (for *m*-flit packets at maximum) with both routers sending their respective packets out at the same time to each other over the respective unidirectional links connecting them. These links are reserved for the swap by the ACK and are not allocated to any other packets by the switch allocators at the two routers.

A successful swap can only be initiated when all the input buffers of both upstream and downstream routers are occupied. If this condition is false, either the swap request is not sent or it is NACK'd. Other conditions for NACK'd requests are discussed in Section 4.2.2.

Deadlock Resolution Time Trade-off. Since deadlocks are rare [5, 9], our implementation allows only one packet swap in the system at any time. This reduces the number of bactracked packets, reduces complexity and eliminates any race conditions that may arise if the same router is both trying to initiate a swap as an upstream router, and acknowledge a swap request as a downstream router. We can tune the rate of deadlock resolution by tuning the swapPeriod. It is possible to have implementations that allow multiple routers in disjoint parts of the network to perform swaps concurrently, or have implementations that detect deadlocks and perform controlled swaps to resolve it, at the cost of more overhead.

4.2 Selecting the packets to swap

4.2.1 *Selecting the swapFwd packet*

Every router has a *swapPointer*. *swapPointer* is valid when there is packet present in any of its input VCs; it points to that VC. At the onset when there are no packets present in the router, *swapPointer* is invalid. If multiple packets arrive at different input ports of the router in the same cycle, the *swapPointer* becomes valid and randomly points to any of the input VCs containing the recently arrived packets. Conditions for further updating *swapPointer* are detailed in Table 2.

The packet sitting in the *swapPointer* VC is the *swapFwd* packet[2] to be sent to downstream router at the swapCycle of *this* router. The downstream router is chosen by the next hop router in the minimal path to this packet's destination based on the routing algorithm.[3] However, if the packet is due to be ejected, it cannot be the *swapFwd* packet, as that would violate the basic requirement of SWAP to have swapFwd packets always make forward progress towards their destination. Thus, the *swapPointer* will move in a round-robin manner to the next non-empty VC with a packet wishing to use a non-ejection port. If no such VC exists, it becomes invalid.

If a valid swapFwd packet exists, the router initiates a swap by sending a *swap_req* to the downstream router. The *swap_req* carries the VC ID as explained in Section 4.2.2. If it is ACK'd, the swap operation is setup for the next cycle. A full swap operation thus takes 4 cycles: (i) req from upstream, (ii) conditions check at downstream, (iii) ACK, and (iv) swap. These are pipelined; each cycle there can only be one router performing a swap, as mentioned in Section 4.1.

[2]More complex solutions to select swapFwd packets can be devised if QoS is needed.

[3]For example, a fully random routing algorithm might pick the next hop based on some congestion metric such as available credits.

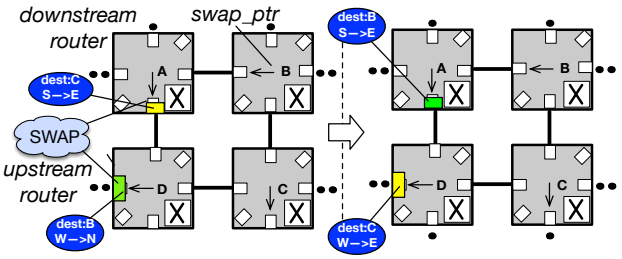

Figure 5: **Example showing that it is possible for both the swapFwd (green) and swapBack (yellow) packets to make forward progress towards their destinations (B and C respectively) after a swap, due to path diversity in the underlying topology**

4.2.2 *Selecting the swapBack packet*

Upon receiving a *swap_req* from an upstream router, the downstream router selects a *swapBack* packet at the input port connected to the upstream router and respond with a *swap_ack*. The swapBack packet is selected from any VC within the protocol message class (inferred from the VC ID sent by the upstream as part of *swap_req*). For simplicity, we select the packet with the same VC ID as the swapFwd packet. Table 2 details the conditions under which the swap is NACK'd (i.e., swap_ack is sent back as 0).

Backtracking. It may appear that the swapBack packet always moves away from its destination. This is not always true. Both the swapFwd and the swapBack packet could move closer to their destinations (i.e., make forward progress), as shown in Fig. 5, due to path diversity in the system.

U-turns. A swapBack packet effectively makes a u-turn, and will request to go back to the router it was swapped from (unless the routing algorithm finds an alternate minimal path for it). After the swap, it will move again either via regular switch allocation or via a swap (once it becomes the swapFwd packet). It is possible for it to backtrack again in the next *swapPeriod* without moving forward due to either of these conditions. But this will never happen indefinitely, as proven in Section 3.3.

4.3 Router microarchitecture

Fig. 6 shows the microarchitecture of the SWAP router. We show a mesh router for simplicity, though the same idea works for a router with any number of ports.

Datapath. We assume bi-directional links. A swap operation requires both a forward path and a backward path to be setup between the upstream and downstream routers (red and blue paths in the Fig. 1) to swap the swapFwd and swapBack packets between their respective VCs. The additions to the conventional router are quite minimal: one 2:1 mux and one 2:1 demux in front of every input port, u-turn support in the crossbar, and a bus connecting all input ports.

As an example, suppose the swapFwd packet is at the South input port Router A, and the swapBack packet is at the West input port at Router B (Fig. 3(a)). For generality, suppose that their current VC IDs are #1 and #3, respectively

Table 2: **SWAP Operation Details.**

Updating swapPointer	Conditions for Failed Swaps
* When the packet pointed by *swapPointer* leaves the router naturally by winning switch arbitration, the swapPointer moves in round-robin fashion to the next non-empty VC. * When a swapFwd packet arrives at this router from an upstream router via a swap. This packet now becomes the swapFwd packet to give it the highest priority at the next swapCycle in case it does not leave naturally.	* At least one of the VCs within the virtual network of the *swap_req* is empty. In this case, the packet could arrive by normal means, and a swap is not required. * In virtual cut-through routers, if the candidate swapFwd and swapBack packet is distributed across two routers, a swap is not performed. In wormhole, this condition leads to packet truncation [7, 8].

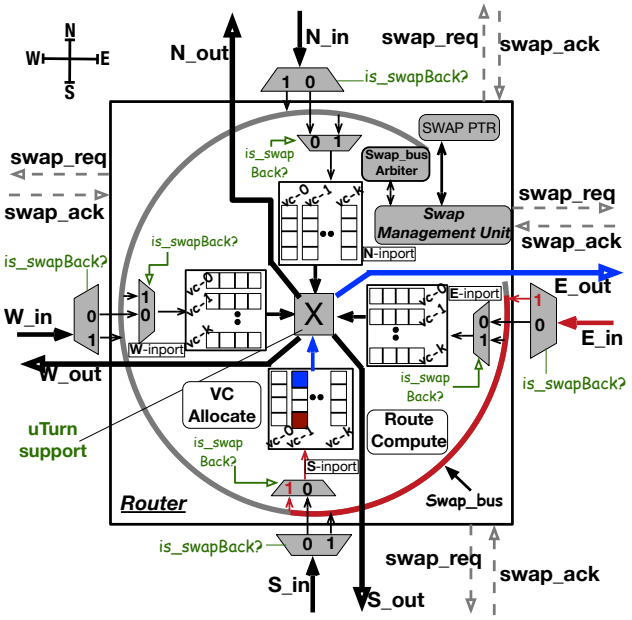

Figure 6: **SWAP Router Microarchitecture. Features added by SWAP are shaded in grey. Datapath: bus connecting all input ports to allow a swapBack packet from the downstream router to get buffered at any input VC, and u-turn support in the crossbar. Control path: Swap Management Unit controlling when and what to swap. The blue and red paths show a swapFwd packet going from South in port to East out port, and a corresponding swapBack packet entering from East out port and getting buffered in the South in port.**

(even though our implementation restricts the swaps to occur within the same VC ID).

- **forward path (A_South_VC_1 to B_West_VC_3):** the swapFwd packet reuses A's crossbar to traverse to B's West input port (blue path in Fig. 6) and gets buffered into VC_3. This is exactly like a regular traversal. As mentioned earlier, during the swap, the output link from A to B is not allocated to any other packet.
- **backward path (B_West_VC_3 to A_South_VC_1):** the swapBack packet reuses B's crossbar to make a u-turn towards A. The swapBack packet arrives at the *East* input port at A, but needs to be buffered at the South input port. The pre-set *swap_bus* transports the packet from East to South, and buffers it in VC_1 (red path in Fig. 6).

Why is a simple bus sufficient? SWAP does not support multiple swaps in the same cycle. Thus, we do not need a crossbar at the input of the router to support multiple swaps from multiple downstream routers simultaneously. The *swap_bus* is pre-configured by the *swap_ack*. This makes the SWAP implementation extremely light-weight.

Table 3: **SWAP vs. Deflection Routing**

	Deflection Routing	SWAP
Mis-routing	Forces packet deflections upon buffers overflow [10] (every cycle in case of bufferless designs [7]) without support for stalls. This leads to high misrouting and congestion.	Provides localized misrouting, which we call *backtracking*, the rate of which can be controlled using the *swapPeriod* parameter.
Spread	Deflections in one part of the network can trigger deflections in another part, leading to high latencies and dropped throughput for all packets.	Backtracking is controlled by swapPeriod and swapCycle parameters.
Router Ports	Indirect restriction on the router micro-architecture: number of input ports must equal the number of output ports of the router.	Places no restrictions on the router's radix, making it more amenable to arbitrary irregular topologies.
Router Critical Path	High hardware overhead for switch-arbiter to perform the best matching upon packet conflict. This also lies in the critical path of the router.	Adds minimal changes to the baseline router micro-architecture (Fig. 6), with the SMU operating off the critical path.
Routing	Deflection routing algorithm is a de-facto routing algorithm, controlled purely by current network congestion	Any routing algorithm (minimal/non-minimal/adaptive) which by itself may or may not be deadlock-prone.

Table 4: **SWAP vs. SPIN**

	SPIN	SWAP
Detection Approach	Maps entire deadlock path upon a timeout using probes that take multiple cycles.	No detection. Performs swaps periodically based on a VC occupancy threshold.
Detection Time	Longer deadlock cycles take longer to map and resolve	Independent of deadlock cycle length
Synchronization	Global: all routers in deadlock must spin at same time	Local: with neighbor who will be performing swap
Resolution Approach	All packets in deadlocked ring move forward simultaneously	Only two packets move simultaneously.
Resolution Time	(N-1) spins in worst case for deadlock of length N	(N-1) swaps of specific packet in worst case for deadlock of length N
Misrouting	None	Backtracks packet one hop

Virtual Cut-Through (VCT) and Wormhole Implementations. VCT routers have buffers deep enough to hold an entire packet. This design naturally works well for SWAP. Swaps are only performed once the entire packet is received, as shown in Table 2. To support SWAP with wormhole routers, we would add packet truncation support, similar to prior works in deflection routers [7, 8, 10]. Packet truncation occurs on swapFwd and/or swapBack packets if the former initiates a swap.

Multi-flit Packets For *m*-flit packets, the swap operation takes *m* cycles, as the flits are serially swapped.

Control Path. The control path of SWAP adds a Swap Management Unit (SMU) that handles if, when and what to swap, as described in Section 4.2.

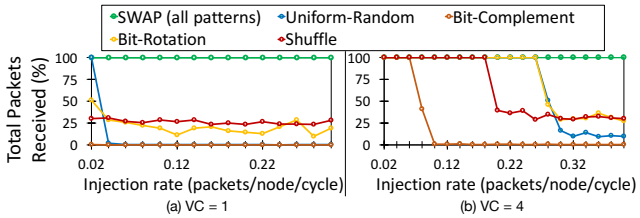

Figure 7: **Percentage of received packets when running a *fully random* routing algorithm. SWAP delivers all packets, irrespective of the traffic pattern. Without SWAP all traffic patterns see a sharp drop in delivered packets, due to deadlocks. The injection rate when deadlocks start depends on the traffic pattern and number of VCs.**

4.4 SWAP vs. Deflection Routing and SPIN

Deflection routing [7], SPIN [9] and SWAP all rely on moving packets in the absence of credits; thus, they are similar in their underlying mechanism for deadlock freedom. Fig. 2 shows an example comparing the three schemes. The key qualitative differences of SWAP versus these are highlighted in Table 3 and Table 4. Quantitative comparisons are present in Section 5.

5. EVALUATION

5.1 Methodology

We use gem5 [36]; to model networks with different configurations we use Garnet [37]. Table 5 provides the detailed configuration parameters. Our baselines include a mix of state-of-the-art deadlock avoidance (deterministic XY, congestion-aware adaptive west-first, and escape VC), recovery (StaticBubble [5], SPIN [9]), and deflection (CHIPPER [8] and MinBD [10]) techniques. Static Bubble relies on extra buffers added in a subset of routers at design time, which are turned on upon deadlock detection (via probes) to drain deadlocked packets. SPIN sends probes (upon timeouts) to detect the deadlock dependence ring, and then performs a coordinated forward movement of the entire ring. For full-system simulations, we run PARSEC [38] and LIGRA [39] (a graph processing suite) over gem5's x86 and RISC-V models which have support for running these respectively.

Irregular Topologies. For our evaluations, we derive some irregular topologies by removing links from a mesh which emulates an SoC with heterogeneous-sized cores or accelerators, or a many core where some links are faulty [16, 17], or have been power-gated [18]. In these scenarios, the resulting topology will not longer be able to use a simple turn-model (e.g., XY/west-first) since certain turns are inevitable to reach some of the destinations. Using these restricted turns can lead to routing deadlocks. We use a spanning tree based Up-Down routing algorithm [16, 17, 40] across all VCs, or within an escape VC, as our baseline deadlock avoidance schemes, and SPIN as the baseline deadlock resolution scheme. For irregular topologies, we assume that information about the missing links and the exact routing path (spanning tree vs minimal) is computed offline and embedded into routing tables [16, 17].

5.2 Correctness

We start by demonstrating why a deadlock-freedom solution is imperative in any network. Fig. 7 runs a set of synthetic traffic patterns with fully-random minimal adaptive routing with no turn restrictions. The occurrence of deadlock causes the percentage of delivered packets to drop sharply; the onset of deadlock depends on the traffic pattern, injection rate, and number of VCs. This shows that deadlocks are highly dependent on the runtime network state. SWAP delivers all packets successfully. To the best of our knowledge, SWAP is the first *non deadlock-recovery-based scheme*[4] to provide fully-adaptive random routing with only 1 VC.

Table 5: **Network Configuration.**

Network	
Topology	8x8 Mesh, Irregular
Routing	Fully-Adapt Random (except when specified)
Latency	Router: 1-cycle, Link: 1-cycle
Num VCs	1, 2, 3, 4
Buffer Organization	Virtual Cut Through Single packet per virtual channel
Deadlock Freedom Mechanism	
Deadlock Avoidance	Mesh: XY, West-first, EscapeVC. Irregular: Up-Down [40]
Deadlock Recovery	Static Bubble [5], SPIN [9] with deadlock-detection threshold=128 cycles
Deflection	CHIPPER [8], MinBD [10]
SWAP	SWAP-K, where K = swapDutyCycle
Traffic Pattern	
Synthetic	Bit-Rotation, Bit-Reverse, Uniform-Random, Transpose, Shuffle. Mix of 1 and 5-flit packets
Real Applications	PARSEC [38], LIGRA [39]
System Configuration (for Real Apps)	
Core	64 cores, x86/RISC-V In-Order, Private L1D=32kB, L1I=32kB, Shared L2 (LLC) Slice=128kB
Memory	MOESI Directory Coherence, 4 DRAM Ctrlrs

5.3 Performance

Synthetic Benchmarks on a Mesh. Fig. 8 shows the performance improvement of SWAP over state-of-the-art deadlock avoidance and recovery schemes on a 8×8 mesh. All designs except XY use adaptive routing. SWAP consistently matches or beats SPIN. Here are some key observations:

- The swapDutyCycle K does not affect the achieved throughput; backtracking does not adversely affect throughput.
- SWAP has a large throughput advantages over CHIPPER, which is known to have low throughput due to deflections. Compared to MinBD, SWAP still provides better throughput because even in the extreme design point of K=1, there is only one potential backtracking packet every cycle (at high-loads), while MinBD at high loads (and high congestion) will result in heavy deflections once its extra buffer becomes full. We quantify this in Fig. 14.
- Compared to avoidance schemes, the performance benefits of SWAP come because it can use fully adaptive random routing, with no turn restrictions. In addition, with SWAP packets can swap and leave a congested region, without relying in credit flow. Both these features push throughput.
- Compared to the recovery schemes (SPIN and Static Bubble), SWAP has no inherent advantages due to path diversity – all designs use fully adaptive random routing. However, the reason SWAP ends up beating SPIN for a few patterns is because once deadlocks kick-in (see Fig. 7),

[4]SPIN [9] is the first to provide fully random routing with only 1 VC but relies on deadlock detection and recovery.

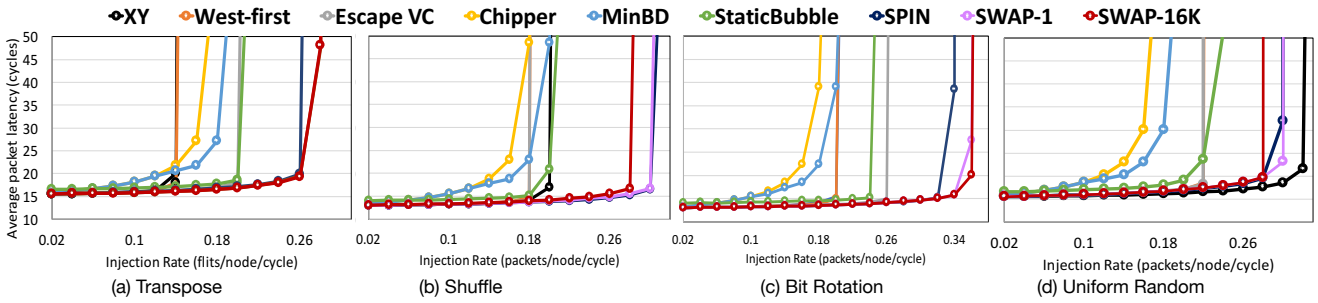

Figure 8: **Performance of SWAP-K (K = swapDutyCycle) with different traffic synthetic patterns, across deadlock-freedom techniques in a 8×8 Mesh. Num VCs=4. Packet Size = Mix of 1 and 4 flits.**

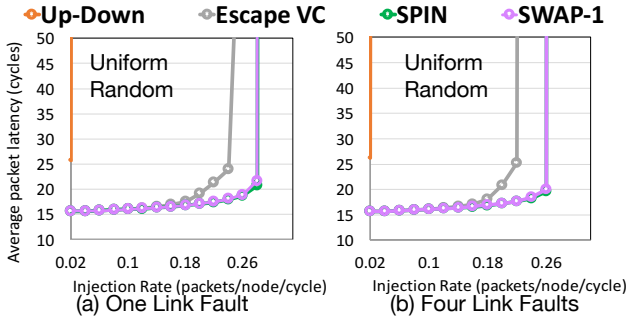

Figure 9: **Performance of deadlock-free networks over Irregular Topologies.**

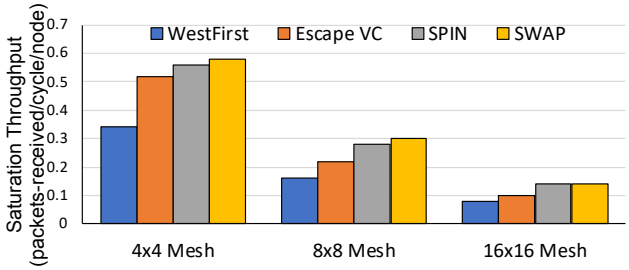

Figure 10: **Effect on throughput as network size increases for Transpose traffic.**

it takes multiple cycles to map out the deadlock path (scales with length of deadlock) and synchronize [5, 9], during which time the network essentially saturates (due to deadlock-driven congestion), leading to loss in throughput. SWAP on the other hand, with periodic swaps with a neighbor, ensures that even if a cycle were to form, it very quickly gets removed.

In summary, SWAP provides robust throughput improvements over both avoidance and resolution-based deadlock-freedom techniques.

Synthetic Benchmarks with Irregular Topologies. Next, we evaluate SWAP with two irregular topologies and compare it against Up-Down routing (i.e., deadlock avoidance), and SPIN (i.e., deadlock resolution). These topologies have one and four links removed in an underlying 8×8 mesh. Fig. 9 plots the latency vs. injection rate for uniform random and shuffle. For irregular topologies, non-minimal routing in Up-Down completely kills throughput. Escape VCs help get some of the throughput back, but still use Up-Down within the escape VC which limits throughput. SWAP gets the same performance as SPIN. *In summary, the performance benefits of SWAP compared to deadlock-free routing algorithms such as Up-Down are magnified when path diversity is at a premium, such as in irregular topologies. Moreover, SWAP matches SPIN without requiring any of the expensive circuitry and signaling overheads for mapping the deadlock cycle and performing global synchronization.*

Scalability study on saturation throughput. We compared the effect on saturation throughput as network size increases across state-of-the-art deadlock avoidance and deadlock recovery schemes in Fig. 10. The analysis is done on the regular mesh, each input port has four VCs in the router (same as Fig. 8). *In summary, we observe the trends in performance remain consistent - SWAP continues to provide higher throughput. However the performance difference between schemes decreases as network size increases.*

Real Benchmarks. Fig. 11 compare the normalized runtime of PARSEC and LIGRA across deadlock-freedom schemes. Real applications do not significantly stress the NoC due to low injection rates; for most benchmarks, all deadlock-freedom schemes fared similarly as the injection rates were quite low. In PARSEC, SWAP shows 30% runtime reduction for swaptions where we saw significant network traffic, and for LIGRA, SWAP provides 2-4% runtime reduction. *This study re-iterates the motivation of deadlocks being rare and probabilistic events, where-in a deadlock-freedom solution is necessary for correctness. SWAP provides deadlock freedom without conservative restrictions for a rare event, or expensive circuitry to detect this event.*

SWAP as an overlay over deadlock-free routing. SWAP can be overlaid on any network since its basic functionality is to enable packet swaps between neighbors. So far we have focused on the deadlock-freedom capabilities of SWAP, but it can also help reduce congestion due to forced forward progress. We ran SWAP with an underlying west-first deadlock-free algorithm. Fig. 12 plots the peak throughput for two synthetic benchmarks, with one and four VCs, normalized to the throughput of a west-first system without SWAP. With one VC, SWAP-1 provides a 12% throughput boost for uniform random, and 6% for bit complement. This result can be interpreted as follows: packet swaps emulate the behavior of additional VCs, as they can force packet movement even if downstream packets are head-of-line blocked. With larger values of K and with higher number of VCs, west-first provides similar performance with and without SWAP.

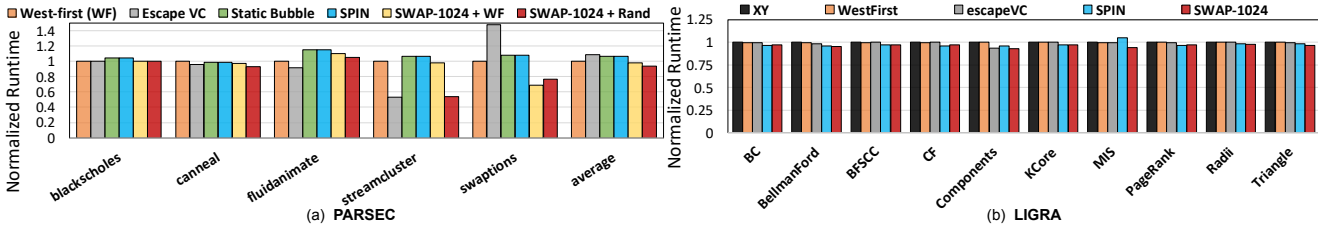

Figure 11: **Normalized Runtime with Multi-threaded Workloads.**

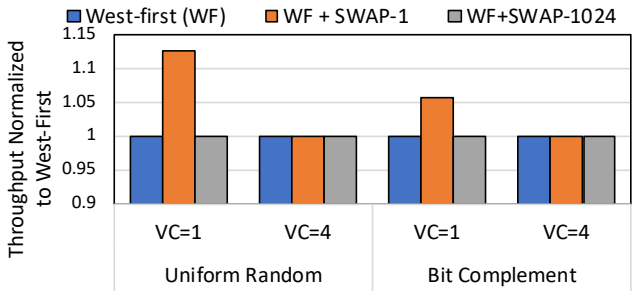

Figure 12: **SWAP throughput with Uniform Random and Bit Complement traffic running with a deadlock free routing algorithm. SWAP provides throughput benefits, especially at low VC counts, by providing extra path diversity. With high VC counts, it is no worse than the underlying algorithm.**

Thus SWAP's backtracking does not adversely affect the underlying system performance. *This study demonstrates that deadlock-freedom benefits aside, adding SWAP to existing routers does not hurt, and can potentially enhance throughput as it emulates the behavior of VCs.*

5.4 Sensitivity Studies

Recall that the swapDutyCycle K controls the rate of swaps in the network. In Fig. 13, we study the impact of K on the number of initiated and successfully executed swaps. Let us start with a pathological worst case: very high post saturation injection rate (last column), and K=1. K=1 leads to a swap getting initiated every cycle by each router in the network in a round-robin manner. In this case, the number of initiated swaps is close to 1, irrespective of VC count since all VCs are active post saturation. Moreover, nearly all swap requests are successfully executed. For the same K=1, when network the injection rate is lower, or if the number of VCs is high, the number of swaps initiated drops close to zero, since the likelihood of the VC pointed to by the swapPointer being empty becomes very high. At very low injection rates (first column), the average number of swaps initiated per cycle is less than 0.02 for VC=1, and less than 0.001 for VC=4. Moreover, the number of successful swaps is zero, since the input port from where a swapBack packet would have been selected has empty VCs. When K becomes greater than 32, the number of initiated swaps drops close to zero at all injection rates. This study shows that the number of swaps is actually quite low; SWAP is a low-complexity solution for deadlock freedom with low overhead. We quantify this overhead next.

5.5 Overheads

Energy Overhead due to Swaps vs. Spins vs. Deflections. Recall that a swap operation involved two packets: a swapFwd and a swapBack packet. The swapFwd packet makes *forward progress*: it gets read out of its upstream buffer, traverses the link, and gets written into the downstream buffer. These are actions it would have had to take anyway and thus do not contribute any energy overhead. The swapBack packet, however, makes a u-turn and goes back to the router it came from. Its corresponding buffer read, link traversal and buffer write are direct energy overheads. Fig. 14 quantifies the extra link activity due to swaps for uniform random traffic. With VC=4, the overhead is close to zero with K=1024, since swap requests fail due to free VCs. With an aggressive K=1, the additional link activity starts rising and goes up to 30% post saturation.

In contrast, CHIPPER and minBD start showing higher link activity at the onset of contention even at low loads, and have 40-80% higher link utilization post saturation. Although SPIN does not inherently misroute like deflection routing or backtrack like SWAP, it adds link activity overhead due to its global synchronization messages (probe, move). This leads to 2.8× higher link activity in the network post saturation due to the incessant number probes that are sent and forked along the way to detect deadlocks.

The link activity behaviour at the extreme design point of one VC is interesting. Here, SPIN's probes increase link activity by 8-10×. Deflection routing NoCs (which do not have VCs) have the same 40-80% higher link activity discussed above. SWAP-1024 sees increased activity, but it remains within 10%. The aggressive K=1 configuration sees a jump in link activity once the network starts to saturate, and eventually adds similar energy overhead as MinBD. *This analysis shows that SWAP is a better design choice than both deflection routing and SPIN. Compared to Deflection routing, it provides steady movement that resolves deadlocks, without adding significant energy overhead due to backtracking as its rate can be controlled. It also provides much better energy efficiency than SPIN as it does not require probe broadcasts which consume significant energy.*

Area Overhead. Fig. 15 plots the area breakdown of the SWAP router compared to XY/West-first, Escape VC, MinBD and SPIN. All routers were implemented using open-source RTL [41] and synthesized and placed-and-routed using TSMC 28nm, targeting 1GHz with 1-cycle pipelines. MinBD, which evolves a bufferless NoC with some buffers, naturally has the lowest area, but comes with misrouting overheads discussed above. All buffered NoCs have 1 VC per port. The escape VC router is assumed to have 2 VCs per port. SPIN's overhead comes due to the synchronization, storage and management of the detected deadlocked loop, that adds about 15% overhead. SWAP has ∼30% lower area overhead than an escape VC router, and ∼4% higher area than a XY/West-first router.

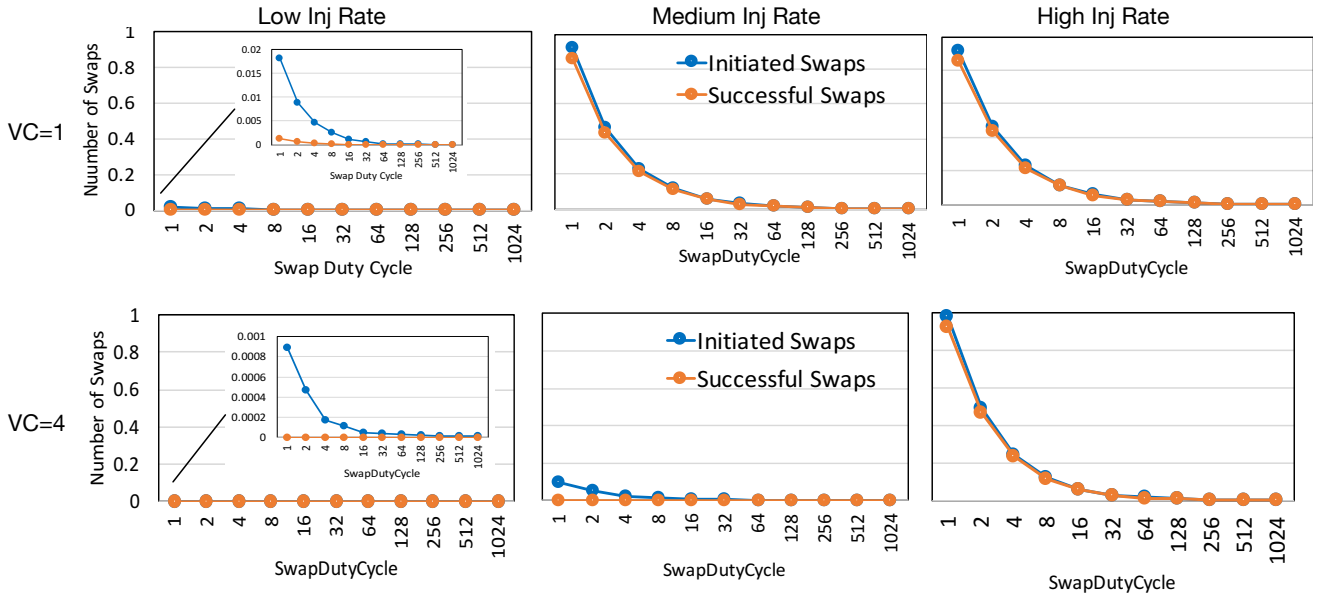

Figure 13: **Relation between number of initiated and successful swaps per cycle, as a function of SwapDutyCycle for low, medium and high injection rates with uniform random traffic. The top row is for VC=1 and the bottom for VC=4. The conditions for unsuccessful (failed) swaps are discussed in Table 2.**

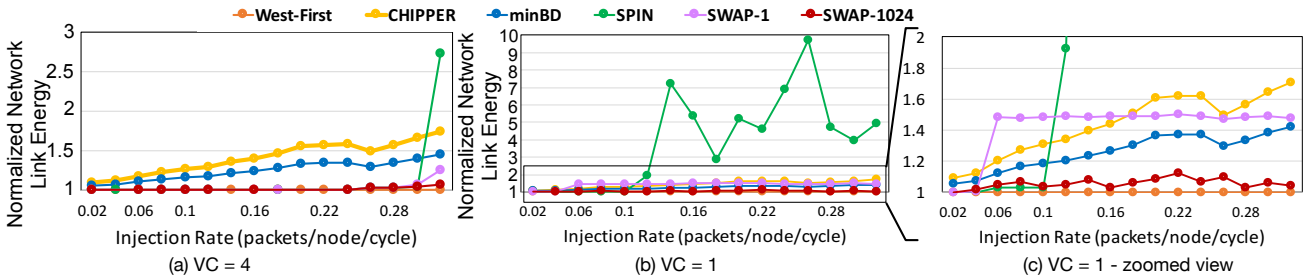

Figure 14: **Energy (i.e., activity) of links in Deflection, SPIN and SWAP networks with VC=4 and VC=1, normalized to a west-first routing algorithm which as purely minimal routing. SWAP's duty cycle parameter can help limit the amount of backtracking.**

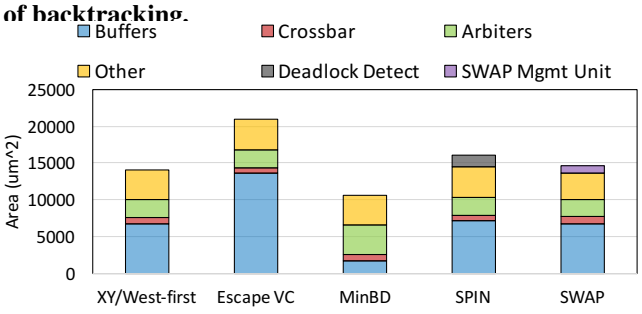

Figure 15: **Post Place-and-Route Router Area (28nm TSMC, 1GHz).**

The area overhead comes due to the swap management components that were presented earlier in Fig. 6.

6. CONCLUSION

We present *SWAP*, a novel technique that enables in-place packet swaps across neighboring routers. SWAP guarantees deadlock freedom by design as it can dynamically break any buffer dependence cycles that might form in the network. It enables the network to take full advantage of available path diversity without requiring turn restrictions, injection restrictions or escape VCs to avoid deadlocks. A steady rate of packet swaps also means that deadlocks do not need to be explicitly detected and recovered from. We present lightweight extensions to implement swaps in our baseline router microarchitecture using a simple bus connecting all input ports of the router, and a unit to initiate the swap at a fixed rate. SWAP is the first non-recovery based deadlock-freedom technique that enables fully-random minimally adaptive routing with just 1 VC. Moreover, it works seamlessly in systems with irregular network topologies, emanating from heterogeneity, faults or power gating. SWAP is a powerful idea that goes beyond just deadlock resolution. It allows packets to escape congested parts of the network and can be overlaid on any network for enhancing throughput, without adding additional buffers.

Acknowledgements

We thank the anonymous reviewers for their helpful comments to improve this work. We thank the Static Bubble and SPIN authors for sharing their gem5 implementations. The deflection code used was implemented by Adarsh Nallamur Krishnakumar, MS (2018) in ECE, Georgia Tech. This work was supported by the National Science Foundation through grants FoMR-1823403, I/UCRC-1439722 and CRII-1755876.

7. REFERENCES

[1] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, pp. 547–553, May 1987.

[2] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 1320–1331, Dec. 1993.

[3] C. Carrion, R. Beivide, J. A. Gregorio, and F. Vallejo, "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*, pp. 322–329, 1997.

[4] V. Puente, C. Izu, R. Beivide, J. Gregorio, F. Vallejo, and J. Prellezo, "The adaptive bubble router," *J. Parallel Distrib. Comput.*, vol. 61, pp. 1180–1208, Sept. 2001.

[5] A. Ramrakhyani and T. Krishna, "Static bubble: A framework for deadlock-free irregular on-chip topologies," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 253–264, 2017.

[6] K. V. Anjan and T. M. Pinkston, "An efficient, fully adaptive deadlock recovery scheme: DISHA," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture, (ISCA)*, pp. 201–210, 1995.

[7] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 196–207, 2009.

[8] C. Fallin, C. Craik, and O. Mutlu, "CHIPPER: A low-complexity bufferless deflection router," in *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture*, pp. 144–155, 2011.

[9] A. Ramrakhyani, P. V. Gratz, and T. Krishna, "Synchronized progress in interconnection networks (SPIN) : A new theory for deadlock freedom," in *International Symposium on Computer Architecture ISCA*, 2018.

[10] C. Fallin, G. Nazario, X. Yu, K. K. Chang, R. Ausavarungnirun, and O. Mutlu, "MinBD: Minimally-buffered deflection routing for energy-efficient interconnect," in *2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pp. 1–10, 2012.

[11] L.-S. Peh and W. J. Dally, "Flit-reservation flow control," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, pp. 73–84, 2000.

[12] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, pp. 874–902, Sept. 1994.

[13] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, pp. 77–88, 2008.

[14] M. Ebrahimi and M. Daneshtalab, "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.

[15] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "ARIADNE: agnostic reconfiguration in a disconnected network environment," in *International Conference on Parallel Architectures and Compilation Techniques, PACT*, pp. 298–309, 2011.

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

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

[18] R. Parikh, R. Das, and V. Bertacco, "Power-aware NoCs through routing and topology reconfiguration," in *Design Automation Conference (DAC)*, 2014.

[19] L. Chen, R. Wang, and T. M. Pinkston, "Critical bubble scheme: An efficient implementation of globally aware network flow control," in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, pp. 592–603, 2011.

[20] L. Chen and T. M. Pinkston, "Worm-bubble flow control," in *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 366–377, 2013.

[21] C. Xiao, M. Zhang, Y. Dou, and Z. Zhao, "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 (EUC*, pp. 353–358, 2008.

[22] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodriguez, J. Labarta, and C. Minkenberg, "On-the-fly adaptive routing in high-radix hierarchical networks," in *Proceedings of the 41st International Conference on Parallel Processing*, pp. 279–288, 2012.

[23] J. Duato and T. M. Pinkston, "A general theory for deadlock-free adaptive routing using a mixed set of resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, pp. 1219–1235, Dec. 2001.

[24] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, pp. 1055–1067, Oct. 1995.

[25] X. Lin, P. K. McKinley, and L. M. Ni, "The message flow model for routing in wormhole-routed networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 7, pp. 755–760, 1995.

[26] S. Ma, N. Enright Jerger, and Z. Wang, "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*, pp. 413–424, 2011.

[27] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *14th International Sympoisum on High-Performance Computer Architecture (HPCA-14*, pp. 203–214, 2008.

[28] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "ViChaR: A dynamic virtual channel regulator for network-on-chip routers," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 333–346, 2006.

[29] A. Samih, R. Wang, A. Krishna, C. Maciocco, T. C. Tai, and Y. Solihin, "Energy-efficient interconnect via router parking," in *19th IEEE International Symposium on High Performance Computer Architecture, HPCA*, pp. 508–519, 2013.

[30] L. Chen and T. M. Pinkston, "NoRD: Node-router decoupling for effective power-gating of on-chip routers," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012.

[31] P. Baran, "On distributed communication networks," *IEEE Trans. on Communications*, 1964.

[32] S. A. R. Jafri, Y.-J. Hong, M. Thottethodi, and T. N. Vijaykumar, "Adaptive flow control for robust performance and energy," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 433–444, 2010.

[33] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, "Evaluating bufferless flow control for on-chip networks," in *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pp. 9–16, IEEE Computer Society, 2010.

[34] P. Lopez, J. M. Martinez, and J. Duato, "A very efficient distributed deadlock detection mechanism for wormhole networks," in *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pp. 57–66, 1998.

[35] Y. H. Song and T. M. Pinkston, "A new mechanism for congestion and deadlock resolution," in *Proceedings of the International Conference on Parallel Processing*, pp. 81–, 2002.

[36] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

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

[38] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[39] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM*

SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 135–146, 2013.

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

[41] 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*, 2017.