SEEC: Stochastic Escape Express Channel

Mayank Parasar
Georgia Institute of Technology
mparasar3@gatech.edu

Natalie Enright Jerger
University of Toronto
enright@ece.utoronto.ca

Paul V. Gratz
Texas A & M
pgratz@tamu.edu

Joshua San Miguel
University of Wisconsin–Madison
jsanmiguel@wisc.edu

Tushar Krishna
Georgia Institute of Technology
tushar@ece.gatech.edu

ABSTRACT

Allocating a free buffer before moving to the next router is a fundamental tenet for packet movement in NoCs. Often, to solve head of line blocking and avoid deadlock, NoCs are provisioned with significant buffer resources in the form of virtual channels (VC) which consume area and power. We introduce stochastic escape express channels (SEEC) to enhance performance and avoid deadlock with dramatically fewer buffers than state-of-the-art NoCs. The network interfaces in SEEC periodically send special tokens called seekers to find packets destined for them and upgrade them to use a novel flow control called Free-Flow (FF). FF-packets traverse the network minimally from link to link, bypassing routers (bufferlessly) to the destination. As a result, FF-packets bypass regions of congestion in the NoC without needing more buffers. Furthermore, any deadlock that a FF-packet was originally involved in is guaranteed to break, without requiring turn restrictions or extra VCs. We also present an extension called multi-SEEC (mSEEC) that enables multiple simultaneous non-intersecting FF-packet traversals to enhance throughput further. We implement and evaluate SEEC and mSEEC on a mesh over a range of synthetic workloads and real applications and observe 34-40% reduction in average packet latency for real applications and 10-50% average improvement in throughput for synthetic traffic over the state-of-the-art at $1/6^{th}$ the area/power budget.

ACM Reference Format:
Mayank Parasar, Natalie Enright Jerger, Paul V. Gratz, Joshua San Miguel, and Tushar Krishna. 2021. SEEC: Stochastic Escape Express Channel. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3458817.3476140

1 INTRODUCTION

An efficiently functioning Network-on-Chip (NoC) is the cornerstone for performance in many-core architectures. Packets' forward progress fundamentally depends on buffer availability. This leads to two sets of challenges that all modern NoCs face.

The first challenge, related to performance, is that of head-of-line (HoL) blocking [29]. Since NoC links are shared, packets can get stuck in congested parts of the NoC, even if their own destinations are not congested. Fig. 1(a) shows an example. Here a packet going towards the *up* region of the NoC is stuck behind those going towards *down (dn)*. Since the *dn* region is congested, the *up* packet cannot move forward, leading to higher latencies and lower throughput.

The second challenge, related to correctness, is that of deadlocks. With full path diversity, packets in different routers can be involved in a cyclic buffer dependency such that they are waiting on the next packet to move to make forward progress in a cyclic manner leading to routing deadlock. Another kind of deadlock is protocol-level deadlock. Protocol deadlock occurs when a packet from one message class is indefinitely blocked by packets from other message classes. The blocked packet cannot move forward because all network buffers are consumed by packets from other message classes.

In current systems, these two challenges are resolved by using Virtual Channels (VCs). To avoid routing deadlocks, turn restrictions are placed on certain VCs (called escape VCs [14]) to ensure that they do not form a cyclic dependence. Alternate techniques to address routing deadlocks are discussed in §2. Protocol deadlocks are resolved by partitioning VCs for each message class (called virtual networks) so that packets from one cannot block the other. Head-of-line blocking is mitigated (though cannot be completely eliminated) by adding more VCs, so that some VCs are free to route traffic to uncongested regions even if other VCs are blocked. The solution of adding VCs to avoid deadlocks and enhance throughput unfortunately comes with high area and power overheads [28, 35]. Moreover, many of these VCs (especially VCs for certain protocol message classes) are highly under-utilized with real-traffic, leading to wasted area and leakage power [28].

We propose SEEC, a novel, unified approach for solving these two sets of challenges. Instead of explicitly adding VCs in every router, SEEC adds a *stochastic escape express channel* that blocked packets can use to make forward progress, without requiring a free buffer at the downstream router. Without adding VCs or buffers, SEEC relies on the observation that *when packets are blocked at routers waiting for credits, the links are still idle*, as shown in Fig. 1(a). SEEC introduces a new flow-control technique called Free Flow (FF). A FF-packet travels *bufferlessly* over the NoC links to its destination network interface (NIC) via a minimal path, bypassing all intermediate routers. A packet is selected to become a FF-packet by a token called a *seeker*. The seeker is sent by a destination NIC to search for any packet in the NoC intended for it, after reserving a buffer slot at the ejection queue. This guarantees ejection for the FF-packet. We also present an enhancement called multi-SEEC

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476140

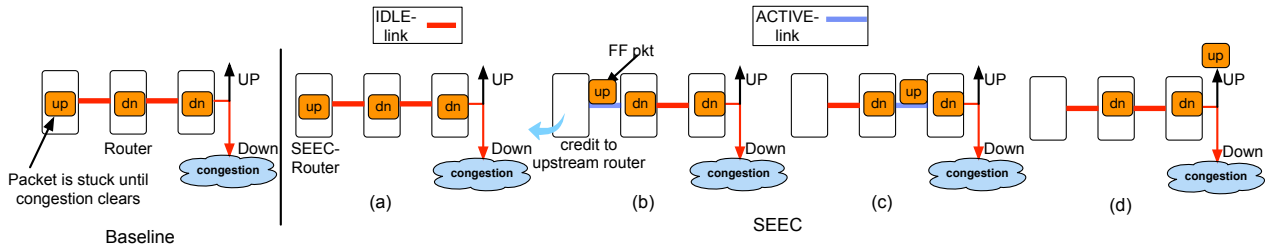

Figure 1: ***Head-of-line Blocking due to congestion. (a) A packet going "up" is blocked by packets going "dn" in the Baseline. (b)-(d) SEEC's FF flow control allows the "up" packet to bypass the congested region to reach its destination.***

(mSEEC) that enables simultaneous minimal bufferless traversals of multiple FF-packets in the NoC with no collisions. Fig. 2 illustrates how FF-packets can resolve routing deadlocks. In Fig. 1, the *up* packet in the SEEC NoC leverages FF flow-control to bypass the congested region to reach its destination.

This work makes the following contributions:

- SEEC, a new flow-control technique that guarantees routing and protocol deadlock freedom and provides higher throughput without requiring any routing restrictions, injection restrictions, escape VCs, virtual networks, or misrouting.
- A novel flow control called Free Flow for bufferless, minimal, non-blocking traversal of packets all the way to their destinations.
- Two policies for upgrading packets to use FF. The base SEEC policy enables only one packet in the NoC to use FF at a time, while mSEEC enables multiple simultaneous non-colliding bufferless FF traversals.
- We implement SEEC and mSEEC on a non-faulty mesh topology and demonstrate 34-40% reduction in average packet latency for real applications and 10-50% average improvement in throughput for synthetic traffic pattern over the state-of-the-art solutions at 1 / 6^{th} area/power budget.

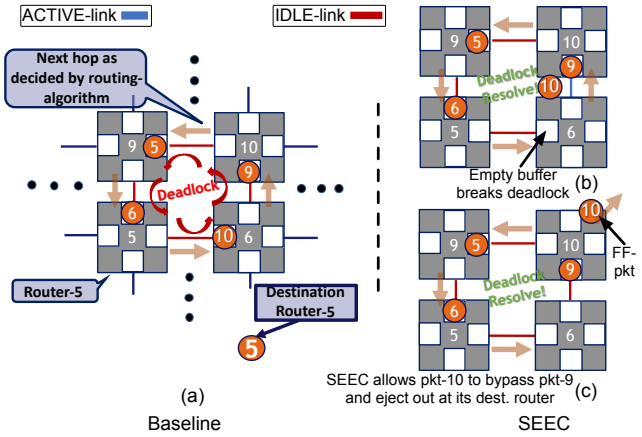

Figure 2: ***Routing Deadlock: (a) Packets' ability to make forward progress is blocked by other packets. Arrows represent desired movement direction. (b) SEEC resolves routing deadlock by allowing FF pkt (#10) to bypass the buffered pkt (#9) until ejection, creating an empty buffer, breaking the deadlock. (c) Shows FF-pkt ejecting out of the NoC.***

2 BACKGROUND AND RELATED WORK

SEEC makes primary contributions in both flow control and deadlock freedom. In this section, we discuss relevant background and contrast related work from both areas.

2.1 Flow-Control Optimizations

In packet-switched NoCs, flow control determines the allocation of flits[1] to input buffers, to ensure flits do not overwrite each other.

Credit Flow Control. Credits are used in the upstream router to track the number of free buffers available at the downstream router to determine if a packet can be forwarded. Inherently, if there are no credits downstream, the packet stalls. To improve NoC performance, past proposals have explored flow control optimizations that create *bufferless bypass paths* in the NoC, i.e., allow packets to only get latched at each hop without any need for buffering and arbitration. In circuit-switched coherence [16], circuits are established on a best-effort basis to allow packets to travel bufferlessly through the NoC. If link resources are not available for reservation, packets start getting buffered again. Express Virtual Channels (EVCs) [22] creates bypass paths of short fixed lengths along the X or Y dimension. Token Flow Control (TFC) [21] leverages tokens, which are hints of credit availability, to create opportunistic bypass paths with turns.

Deflection flow control. An alternate class of solutions has proposed bufferless NoCs [17, 18, 24] where flits do not arbitrate for credits from the downstream router, but instead are deflected in the face of contention (where two flits request the same output port) leading to *misrouting*.

SEEC versus prior Flow Control techniques. Free flow traversal in SEEC (Table 2) is similar in flavor to lookahead-based bypasses in prior work like EVC[22] and TFC[21]. However, while EVC and TFC create opportunistic bypass paths within the NoC, SEEC creates guaranteed bypass paths all the way to the destination NIC. EVCs and TFC also rely on VCs to create bypass paths, unlike SEEC that can work with a single VC. Furthermore, EVC and TFC cannot provide deadlock freedom, relying instead on deadlock-free routing algorithms; SEEC also provides deadlock-freedom, thus supporting any routing algorithm (including fully-adaptive random).

2.2 Deadlock Freedom

Routing deadlocks occur if packets form a dependence cycle in the NoC, as shown in Fig. 2(a). **Protocol deadlocks** can occur if messages from terminating message classes (e.g., responses) in

[1]Packets are broken into one or more flits to match link bandwidth.

Table 1: Qualitative Comparison of Deadlock Freedom Mechanisms. P: Proactive, R: Reactive. S: Subactive.

Techniques↓ / Property →	Full Path Diversity	No Detect deadlock	No Misroute	No Extra Buffers	Routing deadlock freedom	Protocol deadlock freedom
Dally's theory/Acyclic CDG (P) [13]	✗	✓	✓	✓	✓	✗
Duato's theory/Escape VC (P) [14]	✗*	✓	✓	✗	✓	✗
Bubble [8–10, 20, 34, 40] (P)	✓	✓	✗†	✗	✓	✓†
Deflection (P) [17, 18, 24]	✗**	✓	✗	✓	✓	✓
Deadlock Buffers (R) [4, 32, 36, 39]	✓	✗	✓	✗***	✓	✓***
Coordination (R) [35]	✓	✗	✓	✗****	✓	✗
Oblivious Packet Movement (S) [27, 28, 30, 31]	✓	✓	✗	✓	✓	✗/✓††
SEEC (S)	✓	✓	✓	✓	✓	✓

* Within escape VCs: limited path diversity and requires topology information for escape path.
**At low-loads, full path diversity is available. At medium-high loads, packets cannot control the directions or paths along with they are deflected.
***DISHA [4] uses timeout counters present at each input port to choose a packet to eject from the network. It requires a set of extra buffers to route the packet involved in deadlock. Some variations of DISHA, such as mDISHA [39] provide protocol-level deadlock freedom
****SPIN[35] requires a buffer in each router to hold the dynamic deadlock path over which packets involved in deadlock would move synchronously.
† Bubble Coloring [40] provides protocol and routing-level deadlock freedom but requires non-minimal path traversal. Other bubble based schemes only provide routing-deadlock freedom and are minimal or non-minimal depending on the underlying topology.
†† DRAIN [28] provides protocol-deadlock freedom, other prior subactive proposals [27, 30, 31] do not.

coherence protocols are blocked by messages from non-terminating message classes (e.g., requests) [33].

We place routing deadlock-freedom solutions into three categories: proactive (P), reactive (R) and subactive (S), using terminology from DRAIN [28]. Proactive schemes ensure a deadlock is never created in the first place, reactive schemes detect and recover, and subactive schemes allow deadlocks to form but guarantee that they are cleared eventually without requiring explicit detection. Next we discuss the different techniques that address routing deadlocks. Table 1 qualitatively contrasts them against SEEC.

Dally's theory/Acyclic CDG (P). Dally and others [11, 12, 37] prove that applying turn restrictions to packets leads to an acyclic CDG. A key drawback is that turn restrictions reduce path diversity and hurt performance.

Duato's theory/Escape VC (P): These solutions restrict the path that a packet can take only for one VC per input port, called an Escape VC. Packets enjoy full path diversity while in normal VCs [14, 15]. The key challenge is that Escape VCs require an extra VC at each input port for implementation.

Bubble (P): A bubble refers to an empty buffer or VC. The underlying theory [8–10, 20, 34] is that as long as there is an empty buffer present in a *ring*, that ring is deadlock free. The key drawbacks are that this only works on rings and tori and global coordination for tracking the bubble.

Deflection (P): Deflection routing argues that as long as packets keep moving every cycle, either towards or away from their destination, the network is deadlock free. It follows hot-potato routing [5] in which packets get misrouted in the face of contention. The key drawbacks include non-minimal routes, leading to higher latency and dynamic energy, and expensive solutions to *guarantee* livelock freedom [17, 18, 24].

Deadlock Buffer (R): Schemes such as DISHA [4] and its extensions [32, 36, 39] embed extra *deadlock buffers* at design-time in all routers, which remain off during nominal operation. Deadlocks are detected via time-outs and blocked packets are progressively routed to their destinations via these deadlock buffers.

Coordination (R): Coordination schemes (e.g., SPIN [35]) detect deadlock and synchronously move packets to resolve deadlock. The key drawback is the complex deadlock detection circuitry to detect and map the deadlock ring at runtime. Moreover, it requires global synchronization among the packets that must move simultaneously to resolve the deadlock.

Oblivious Packet Movement (S): Recent work proposes a class of solutions which leverage periodic packet movement in the network [26–28, 30, 31] to remove deadlocks. BBR [31] proposes moving packets within a router using bubbles. PitStop [19] temporarily moves packets to empty slots in the NIC. BINDU [30] moves packets between neighboring routers using a bubble along an embedded cyclic path. SWAP [27] proposes *swapping* of packets across neighboring routers while DRAIN [28] proposes movement of packets along an embedded ring across the entire network. The underlying theory is that if packets are periodically forced to move obliviously in the network then any routing deadlock can be resolved. The periodic and oblivious nature of packet movement overlaid on a deadlock prone minimal routing algorithm in these solutions ensures deadlock freedom. A key challenge is that these oblivious packet movements may misroute a packet away from its destination; therefore, a packet may need to traverse the network non-minimally. Additionally in pathological cases, the deadlock may be slow to resolve.

The most common approach for protocol deadlock freedom is separate virtual networks (VNets) [28] in the NoC for each message class. Though some prior works provide protocol deadlock freedom without VNets [19, 28, 39, 40], they come at the cost of misrouting [28] and/or non-minimal routes [39, 40].

SEEC vs. prior Deadlock Freedom techniques. Table 1 contrasts prior works against SEEC. SEEC guarantees both routing and protocol deadlock freedom with the theoretical minimum of one VNet with one VC per input port in the NoC. SEEC is also subactive, i.e., it can allow deadlocks to form but ensures that they will be broken (as packets use FF to exit the NoC) without requiring explicit detection. However, SEEC does not cause misroutes unlike prior subactive schemes.

Table 2: Key Terms in SEEC.

Term	Definition
Free Flow (FF)	A flow control scheme where packets are not buffered, and move forward every cycle, without the need for credits.
Seeker	Token sent by the destination NIC after reserving an ejection VC w/in a specific message class. It searches for a packet in the NoC intended for this NIC in that message class.
Seeker path	A side-band network connecting all routers in the network over which the seeker visits all routers, and comes back to its initiator in case it does not find a packet.
FF packet	This packet is chosen by the seeker and traverses to its destination over a *minimal path* using FF.
Lookahead	A signal sent one hop ahead of FF-packet so that next-hop router prioritizes the movement of incoming FF packet over the normal buffered packet.
mSEEC	An extension to SEEC where multiple seekers are sent out at the same time to different network partitions. The FF packets use non-overlapping, minimal routes.

3 SEEC

We describe SEEC from concept to implementation. Table 2 lists some key terms used throughout the paper. As an analogy for SEEC, consider the flow of traffic; cars and trucks are required to obey traffic signals. A car cannot advance to the next block until the light turns green (this is analogous to waiting for a credit). However, some designated vehicles can ignore such traffic rules. Emergency vehicles are an example; regular cars yield to emergency vehicles and let them pass. Similarly in SEEC, one packet (at a time) in the system gets elevated in priority to use an express path to reach the destination.

3.1 Free Flow

We propose a new flow control scheme called **Free Flow (FF)**. A packet using FF is given priority at each router over regular packets which use credit-based flow control, allowing its flits to traverse the network bufferlessly, over a minimal route all the way to its destination NIC. Unlike deflection flow control (§2.1), there is no misrouting of FF packets. This is ensured by guaranteeing that multiple packets concurrently traversing the NoC using FF use non-intersecting paths. For the base SEEC design, there can only be one FF packet in the network at any given time. §3.8 discusses how multiple FF packets with non-intersecting paths can be guaranteed. Unlike TFC and EVC, SEEC *reserves* the buffer at the destination NIC before upgrading the packet to FF. This frees SEEC from *buffer turn around time* considerations and minimizes the buffering required. This is a unique feature of Free Flow control.

3.2 Overview

In SEEC, all packets nominally use credit flow control. In a round-robin manner, each destination NIC selects a packet in the network destined for itself to use FF. The mechanism for selecting such a packet is discussed in §3.3. This packet traverses the NoC in deterministic time by being prioritized to use the link at every hop along its route, without getting buffered within each intermediate router, only latched at each hop). This is implemented by sending a lookahead signal one cycle ahead on a dedicated link, similar to prior work [21, 22]. The lookahead carries the output port of the FF packet, and the intermediate router sets up its crossbar for the FF packet arriving next cycle, disallowing its own locally buffered flits from using that output link. As a result, the FF packet can bypass both contention and deadlock in the NoC. We describe the detailed operation in the next section.

3.3 Operation Details

System Assumptions. SEEC assumes that the NIC has per-message class ejection VCs, as shown in Fig. 3. However, within the NoC, all messages share the same set of VCs. Separate VNets do not exist.

Mechanism for choosing FF packet and guaranteeing ejection. A NIC initiates a SEEC by reserving an ejection VC within a message class, and circulating a *Seeker* through the NoC over a pre-defined path covering all routers to search for any packet intended for this ejection VC. If the seeker finds such a packet, the packet traverses the NoC via FF and the seeker is dropped. Since an ejection VC is reserved before the seeker starts searching, the FF packet is guaranteed to be consumed. Once the FF packet is ejected, or if the seeker returns, the control moves in a round-robin manner to ejection VCs of the next message class at this router, and then to the neighboring router, and so on, to send out their seekers.

What if the seeker does not find any packet? If the seeker circulates back to the original router, this indicates that there was no packet in this message class in the NoC currently waiting to get routed to this NIC. The reserved ejection VC is freed up and the same process is repeated for the remaining ejection VCs. Once all ejection VCs are exhausted, the router notifies its neighboring router of its turn to send seekers.

What if no ejection VC is free in a message class? In this case, it passes its turn in a round-robin manner to the next message class, and eventually to the neighboring router to send its seeker signal instead. However, once a message class that missed its turn gets a free ejection VC, it is pro-actively reserved for the next round. No packets are allowed to use this ejection VC until this VC's next turn for sending a seeker.

How is the SEEC path implemented? In our implementation, the SEEC path is a ring through all routers in the NoC (as shown in Fig. 3). Each router has local state to store the order in which the seeker should search through all input ports with that router, and where to forward the seeker next. SEEC can be implemented with any seeker path as long as it traverses all routers. For mSEEC (§3.8), the seekers' paths no longer span the entire NoC, but instead only within partitions.

What is the policy for searching for packets within the SEEC path? There could be myriad policies used by the seeker for selecting packets to upgrade to FF. For fairness, we implement a *round-robin* policy, where each destination logs the location of the $< router - id\_inport - id >$ from where the last FF packet was selected, and begins the search from there. It searches through all routers and ports, and returns to its sender if it did not find a packet.

What if the ejection port is busy when the FF packet arrives? Though the FF packet is guaranteed to be ejected, it is possible for it to arrive at its destination router while another multi-flit packet is in the process of getting ejected. The FF packet in this case will take precedence over the ongoing ejection and the current ejection

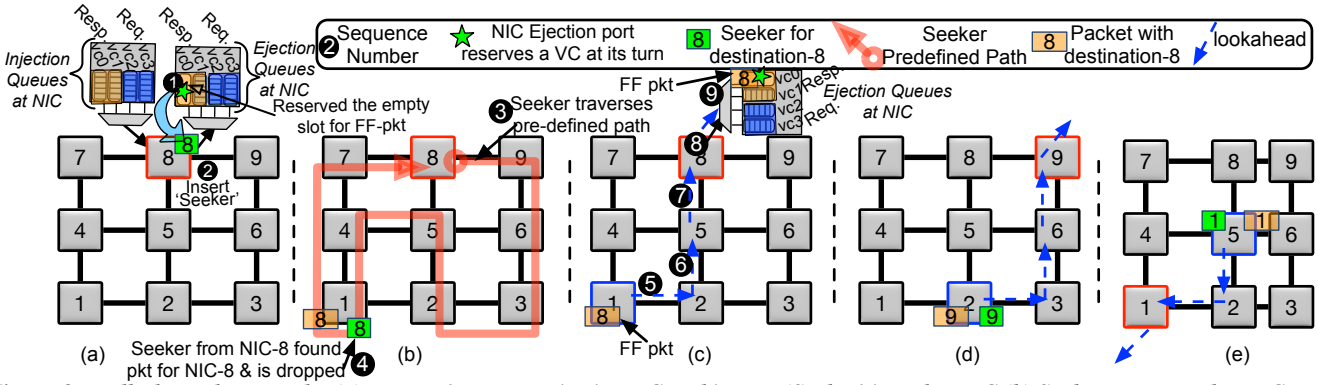

Figure 3: ***Walk-through Example: (a) Router-8 reserves ejection VC and inserts 'Seeker' into the NoC (b) Seeker traverses the NoC on a predefined path to look for packets to eject. (c) Seeker finds a packet at router-1, Seeker is dropped and this buffered packet now becomes FF-packet (d) After router-8, router-9 repeats the process of reserving an ejection VC (not shown to save space); sending seeker and ejecting packet bufferlessly (e) After router-9 (last router), router-1 (first router) repeats the process.***

will wait until the FF-packet gets ejected from the network. Since the ejection VC for the FF packet was reserved prior to the seeker being sent, the stalled ejection will not be for the same VC.

3.4 Walk-through Example

Fig. 3 shows a walk-through example of SEEC. As discussed in §3.3, a packet is chosen to become FF by its destination NIC. This is done with the help of a seeker. In this example, router-8 first reserves an ejection VC in the response message class at NIC-8. It then sends the seeker (Fig. 3(a)) on a predefined path covering all the routers in the NoC at least once. During its network traversal, if the seeker finds a response packet whose destination is router-8, it converts this normal buffered packet into the FF packet. In this example, this packet is at router-1 (Fig. 3(b)). The seeker is dropped and the newly dubbed FF packet travels minimally to its destination via an express path created using lookaheads, as shown in Fig. 3(c). Once this original FF packet is ejected, the same process is repeated for the next message class. After all message classes have tried (successfully or unsuccessfully) to receive a FF packet, the next router is notified to send its seeker. This process repeats in a cyclic fashion over the topology (Fig. 3(d)(e)).

3.5 Lookaheads

To allow the FF packet to make forward progress every cycle, and to suppress the movement of normal packets at the next downstream router through the same output port, a lookahead signal is sent by the router which accepted the seeker to upgrade its buffered packet to FF. This lookahead signal is sent on its dedicated link one cycle ahead of the data packet, similar to prior work [21, 22]. The lookahead reserves the output port for the FF packet at the next downstream router as dictated by the routing algorithm. §3.10 further describes the router micro-architecture modifications over the baseline router to realize a SEEC router.

3.6 Overhead of Sideband Networks: Seeker and Lookahead

Fig. 4 shows the formats of the seeker and lookahead packets. The seeker carries the ID of the destination NIC and message class that injected it. It can also optionally carry the ID of the router from

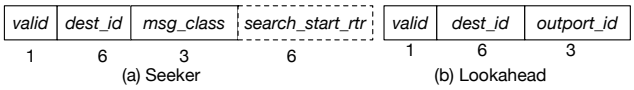

Figure 4: Seeker and Lookahead Packets

where to begin searching (for QoS, so that routers closer to a NIC on the seeker path do not end up always upgrading their packets). This leads to a 10-16 bit uni-directional ring, in a 64-core mesh with 6 message classes. While it is possible to multiplex the seeker over the regular links, prioritizing it over flits, we add a separate sideband path for simplicity. Compared to multiple 128-bit links and flip-flops switching within the NoC every cycle, we found the 16-bit link activity of 1 (SEEC) or 8 (mSEEC) seekers negligible.

The lookahead carries the output port ID and destination ID so that it can pre-compute the output port for the next router [21, 22]. This is 10 bits wide for a 64-core mesh.

3.7 Proof of Correctness

We first prove that SEEC guarantees resolution of protocol deadlocks and then extend the proof for routing deadlocks.

Assumption: The ejection port at the NIC has separate VCs for each message class of the protocol. Within the NoC, SEEC can operate with a single unified VC for all message classes.

Definition: A Terminating Message Class refers to a message class that ends the protocol transaction, e.g., responses.

Necessary condition for breaking protocol deadlocks. Messages belonging to terminating message classes should never be indefinitely blocked within the NoC.

Lemma 1: Messages within the ejection VC for a terminating message class in a coherence protocol are always guaranteed to be consumed by the cache controller (*aka consumption assumption*) and do not block waiting for other messages.

Proof: When a request is issued to the NoC, an entry is allocated in a Miss Status Handling Register (MSHR) and injected into the outgoing request VC at the NIC. The response (data or ACK) arrives in the response VC at the NIC and will be consumed by the reserved MSHR entry for processing. If the rate of egress to the MSHR is slower than the rate of ingress into the ejection VC from the NoC, the

ejection VC may temporarily be full, but will eventually become free to eject new packets from the NoC. Similarly, invalidation requests issued by are replied to with acknowledgment (ack) messages. While invalidation request processing may be blocked by the inability to insert acks, acks are immediately consumed upon receipt at the directory. We cannot list every protocol dependence scenario, but terminating message classes such as responses/ACKs in protocols are built with the consumption assumption [33].

Lemma 2: A response packet causing protocol deadlock is guaranteed to reach its destination via FF to break the deadlock.

Proof: Suppose a response packet is stuck in the NoC behind requests and is part of a protocol deadlock. Thus it cannot make forward progress via regular means. The response message class at its destination will eventually have a free ejection VC for this specific packet as discussed by Lemma 1. Since each message class at each NIC gets a fair chance to send a seeker in a round-robin manner, its destination will eventually send a seeker and enable this packet to unblock.

SEEC ensures that any response blocked indefinitely in the NoC buffers (i.e., protocol deadlock) will eventually be found by the seeker from its destination NIC and drained out, breaking the deadlock. However, theoretically there could be a corner case where all NoC buffers are full of requests and a response can never get injected into the NoC. To account for this, once every N cycles, the seekers from each destination also search the injection buffers at the NIC of each router.[2]

Lemma 1 + Lemma 2 prove that SEEC guarantees protocol-deadlock freedom.

Corollary 1: A request VC at the ejection port will never remain indefinitely blocked.

Proof: Since the network is protocol deadlock free, a request VC will not remain indefinitely blocked waiting for responses. Suppose a request message class does not have a free ejection VC during its turn to send a seeker, SEEC will send a seeker for the next message class. However, Lemma 2 and Corollary 1 prove that *all* message classes will eventually get a free ejection VC. Once the message class that missed its turn gets a free VC, this VC is *proactively reserved*. No packets are allowed to be ejected into this VC until this VC's next turn for sending a seeker. This bounds the number of times a message class might miss its turn to send a seeker.

Lemma 3: Every packet in a routing deadlock is guaranteed to reach its destination via FF control.

Proof: From Lemma 2 and Corollary 1, all message classes will eventually get a free ejection VC and get a chance to send a seeker. This will allow any packets in a cyclic dependence in the network to use FF to reach their destination, breaking the cycle.

Lemma 3 proves SEEC guarantees routing-deadlock freedom.

Livelock Freedom Proof. SEEC is livelock free because all the packets are routed minimally through the network. Thus SEEC does not need any additional livelock handling mechanism like tracking and prioritizing oldest packets [17, 18, 24] or performing network drains [28].

Point-to-Point ordering. Using FF, packets may get re-ordered from a given source to its destination. This effect is not unique to SEEC and is also present in adaptive routing algorithms [38]. Many commercial protocols such as HyperTransport support out-of-order delivery of messages [25]. For the protocols that do not, we assume re-order buffers for those message classes that need point-to-point ordering, and leave point-to-point ordering support for future work.

[2]In our simulations, N was set to 1M cycles, but we never encountered this corner case even with a single VC at every router.

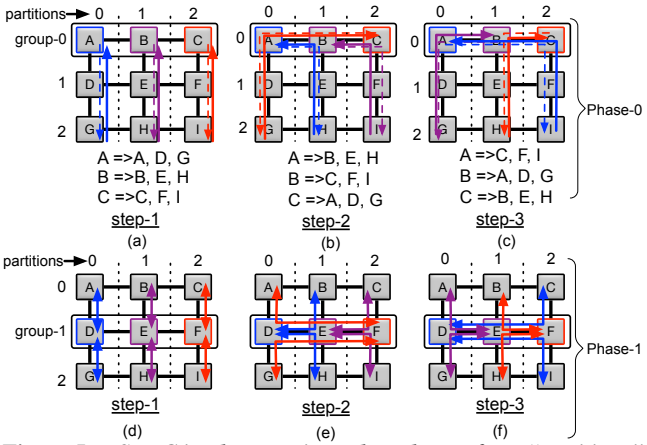

Figure 5: ***mSEEC implementation. The columns form "partitions" and the rows are "groups". In Phase-0, group-0 sends seekers to each partition. Phase-0's NICs send seekers to the routers listed after $'=>'$. Dotted lines represent the seeker path. FF-packet follows the same path in the opposite direction. No two paths overlap. Thus all FF-packets will simultaneously use minimal paths without collisions. In phase-1 double-ended arrows have been shown to convey seeker and FF-packet paths.***

3.8 Multi-SEEC (mSEEC)

In SEEC, we allow sending one seeker at a time on a pre-defined side-band path; as a result, there is a maximum of one FF packet at a time in the NoC. However, many popular network topologies such as meshes have lots of inherent path diversity; sending one seeker at a time under utilizes the available path diversity for creating express paths. This is the motivation behind multi-SEEC (mSEEC).

Abstract Implementation of mSEEC. To implement mSEEC, the topology *N* NICs needs to be divided into *P* independent *partitions*, and the NICs into *N* / *P groups* with *P* NICs each, one from each partition. Each partition receives a seeker from a different NIC. mSEEC operates over multiple phases. In the first phase, the *P* NICs from the first group *simultaneously* search for packets which want to eject out from those partitions, using their respective seekers. Once the search, followed by FF-packet traversal is over (which is bounded due to the fixed time it takes for the seeker to travel to the furthest router in its partition and the FF-packet to come back), the NICs permute these independent partitions among themselves. At the end of each phase, the *P* NICs would have searched through the entire topology. In the next phase, the next group of NICs follow the same approach. To ensure no collisions, the partitions and groups need to be chosen in a way that the paths of the *P* seekers do not overlap during the seek portion, and the paths of the *P* FF-packets do not overlap during the FF portion of each phase.

Our Implementation of mSEEC. We choose the partitions as the columns of the Mesh, and the groups as the rows of the Mesh. Fig. 5 shows an example. In phase 0, during Step 1 (Fig. 5(a)), the three

Table 3: SEEC versus mSEEC on $k \times k$ Mesh, m Msg Classes.

Property	SEEC	mSEEC
Seeker Path	Embedded Ring	Non-overlapping Minimal
Num of simult. seekers	1	k
Seek Time	1 to O($m\times k^2$)	1 to O($m\times k$)
Deadlock Res. Time (worst-case)	O($m\times k^4$)	O($m\times k^3$)
FF Packet Path	Minimal	Minimal

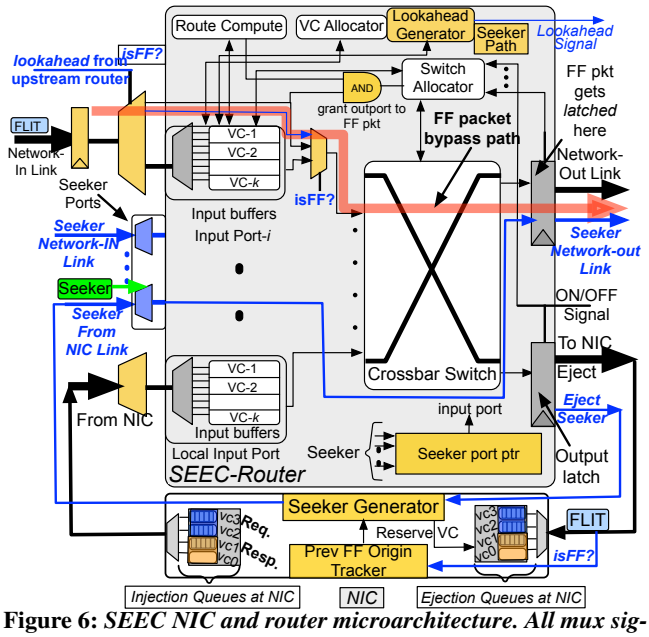

Figure 6: ***SEEC NIC and router microarchitecture. All mux signals are set up by the lookahead signal in advance, this allows seamless traversal of bufferless FF packet.***

NICs (A, B, C) in row 0 of the Mesh simultaneously seek packets within Columns 0, 1 and 2. Next, they seek within Columns 1, 2 and 0, respectively (Fig. 5(b)). Next, in Columns 2, 0, and 1, respectively (Fig. 5(c)). The dotted line represents the path taken by the seekers, and the solid line is the path taken by the FF packets (which just follow the reverse path of the seekers). After the first row finishes seeking packets from all the columns (i.e., the entire topology), in Phase 1, the next row now seeks packets in a round-robin manner over the columns of the topology (Fig. 5(d)-(f)).

Fig. 5 shows that each phase has a fixed number of steps, and in each step, there are fixed cycles involved for it to complete. For example in Phase 0, Step 1, a *seeker* will take at most 2 hops to cover it's own column (partition). However, in Phase 0, Step 2 the *seeker* from router-C's NIC will take 4 hops. This pre-computed static time-bound can be used to schedule each phase to realize mSEEC. We leave the exploration of alternative partitioning schemes for mSEEC to future work.

SEEC vs mSEEC. Table 3 summarizes the key differences between SEEC and mSEEC. Since mSEEC has multiple seekers, each searching through a smaller part of the network, it can resolve deadlocks faster.

3.9 NIC Microarchitecture

The NIC attached to a router inserts seekers into the NoC in a round-robin fashion. SEEC augments the baseline NIC architecture with additional logic to incorporate the following features (shown in Fig. 6).

Seeker Generator. Each message class in the NIC gets the opportunity to generate and inject a seeker, one after the other. As discussed in §3.3, the condition to inject a seeker is after *reserving* an empty ejection VC. If no VC is empty in some message class, the turn moves on to the next message class, and eventually to the next router's NIC. Any time a message class skips its turn due to unavailable VCs, the NIC proactively reserves the VC whenever it becomes available to be used in its next turn. §3.7 discusses the proof for guaranteeing that an empty VC will eventually show up in each message class. The seeker carries the NIC ID and message class within it. It also carries information about the search strategy.

Prev FF Origin Tracker.: Each NIC houses a special register which stores the $< router - id\_inport - id >$ from where the last FF packet was successfully selected for ejection at this NIC. The seeker generated by this NIC would start looking for the packet starting from this $< router - id\_inport - id >$, to maintain a round-robin search policy.

Since the NIC is directly connected to the local port of the router, we include the area and power estimates of these additional components as part of the router for the purposes of comparison against baselines (§4.2).

3.10 Router Microarchitecture

Fig. 6 shows the microarchitecture of a SEEC router. A FF flit entering the router bypasses the router's input port and directly traverse its crossbar (setup in the previous cycle by the lookahead). The FF flit is then latched at the output port, as shown in the figure.

Lookahead signal: A lookahead signal is sent one cycle in advance of the FF packet traversing the NoC to set the mux and prioritization logic appropriately. The lookahead generator reads the header from the input VC for the FF-packet and sends it a cycle-ahead of the payload. Lookaheads are generated by input ports and not by the NICs. At the destination router, these lookaheads are used to temporarily stall the ongoing ejection (if any) to prioritize the ejection of FF-packet.

FF bypass logic: An extra bypass mux is added before the crossbar input which allows the FF packet to bypass the buffers and acquire the output port. The lookahead received at the router overrides the local Switch Allocation grant so that the FF packet gets higher priority.

Seeker port ptr is used to provide the round-robin priority to all the input ports of the router to search for packets to upgrade to FF for the seeker. The search involves a parallel comparison of the message-class-id and dest-id field from the VC state (populated by head flits) of all input VCs in the router. We were achieve to a single cycle latency at 1GHz for this 9-bit comparison; we tested up to 20 VCs (4 per port). Recall that SEEC can provide both protocol and routing deadlock-freedom with a single VC, so additional VCs are only required for performance, not correctness.

Seeker Path is a pointer to the next router the seeker should be routed to depending on the embedded seeker path.

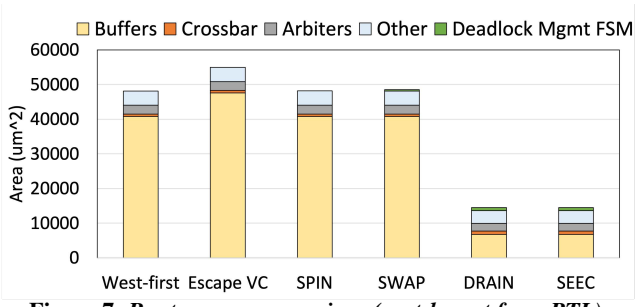

Figure 7: *Router area comparison (post-layout from RTL).*

The signals mentioned in the SEEC router (Fig. 6) are setup by the lookahead signal prior to the arrival of FF packet. The highlighted components in Fig. 6 are unique to SEEC. We quantify the overhead in §4.2.

3.11 SEEC across Buffer Management Schemes

Virtual Cut Through (VCT). In VCT, both buffers and links are allocated at packet granularity. However, the complete packet may not be present in the buffer, but remaining flits would be following the head flit in close succession. The seeker makes the head flit FF and records this information at the router. The remaining flits of the packet that subsequently arrive follow the head using FF.

Wormhole. In wormhole, unlike VCT, buffers may be smaller than the number of flits in the largest packet. Allowing adaptive routing with wormhole flow control adds the constraint that VCs must only contain one packet at a time to avoid deadlocks [13]. This requirement makes it easy to extend SEEC to wormhole networks. The seeker need only examine the flit at the front of a given VC queue, only upgrading it if it is a head flit. If the head is upgraded to FF, the remaining flits of the packet that subsequently arrive in this queue are marked FF as well. This approach will work even if the wormhole queue has the minimum depth of 1-flit. SEEC thus has added advantages over other deadlock-freedom schemes [35],[27],[24] as it does not require packet truncation to support wormhole, nor does it re-order flits within the packet [24].

3.12 SEEC/mSEEC over irregular topologies

SEEC can be implemented on any topology, regular or irregular (*e.g.,* a regular topology with faults) over which a virtual ring can be embedded to create a seeker path. As prior work DRAIN [28] has shown, the three conditions for this are: (i) no network disconnection, (ii) all links are bidirectional and (iii) each router allows U-turns. However, additional complexity will be needed to embed the routing path. For mSEEC, there is added complexity of creating independent, non-overlapping seeker paths which may be harder on arbitrary topologies. SEEC/mSEEC implementation on irregular topologies is beyond the scope of this paper.

4 EVALUATION

4.1 Methodology

We evaluate SEEC using *gem5* [7] with the *Garnet2.0* [3] network model and *Ruby* memory model. We modeled the baseline routers using OpenSMART [23] and obtained area numbers from synthesis and PnR using FreePDK15nm. We compare SEEC and mSEEC against state-of-the-art techniques using both synthetic traffic and full-system applications. Table 4 lists our simulation parameters.

Table 4: Key Simulation Parameters.

Real application simulation parameters	
Core	16 cores; x86 ISA (PARSEC, SPLASH-2), 1 GHz Out of Order cores, No prefetcher
L1 Cache	Private, 32kB Ins., 64kB Data, 4-way set associative
Last Level Cache	Shared, distributed, 2MB, 8-way set associative
Cache Coherence	MOESI, VNet=6
Network parameters	
Topology	4×4, 8×8, 16×16 Mesh (synthetic traffic), 4×4 Mesh (PARSEC and SPLASH-2)
NoC Baselines (*Deadlock-freedom - P,R,S, Routing Algorithm*)	Turn Models (*P, XY/West-first*) MinBD (*P, Deflection*) Token Flow Control (*P, West-first*) Escape VC (*P, Fully adaptive random in regular VCs, West-first in Esc VC*) SPIN (*R, Fully adaptive random*) SWAP/DRAIN/(m)SEEC (*S, Fully adaptive minimal random*)
Router Latency	1-cycle
Virtual Network	6-VNet (Turn Model, Esc VC, TFC, SPIN, SWAP) 1-VNet (DRAIN,SEEC, mSEEC) 2 VCs/VNet
Buffer Org.	Virtual Cut Through, Single packet per VC Mixed 1-flit (request, ack), 5-flit (response) packets
Link Bandwidth	128 bits/cycle

Baselines. We compare SEEC and mSEEC against several baselines from a spectrum of state-of-the-art prior work in deadlock-freedom and flow-control optimizations, as shown in Table 4. The baselines that rely on proactive deadlock-freedom (i.e., XY, West-first, TFC and Escape VC) have routing restrictions, either in all VCs or in their escape VC. Deflection routing (i.e., minBD[18]) leverages minimal routing by default, but misroutes due to deflections can make the route non-minimal. The reactive (i.e., SPIN [35]) and sub-active (i.e., SWAP [27] and DRAIN [28]) approaches leverage *fully-adaptive random minimal* routing to provide full path-diversity to packets (unless explicitly stated otherwise). Adaptive routing uses the number of free VCs at the downstream routers to decide the direction, given a choice.

Workloads. We use both real applications (PARSEC-3.0 [6] and SPLASH-2 [41]) and synthetic traffic. We evaluate synthetic traffic with a mix of 1- and 5-flit packets. The simulator is warmed for 1000 cycles to remove any effects due to empty queues in the packet latency statistics.

4.2 Area

Fig. 7 shows the normalized area breakdown, when compared to Escape VC, SPIN, SWAP and DRAIN. We implement the configuration with the minimum number of buffers required by the router for each scheme for the NoC to function correctly. Thus Escape VC uses 7 VCs (1 VC per VNet + 1 shared VC for adaptive routing), West-first, SPIN and SWAP use 6 VCs (1 VC per VNet), DRAIN and SEEC use 1 VC. The additional components in the NIC for SEEC are included in the router area for the purposes of overhead comparison against prior art.

The major benefit of SEEC is providing both router and protocol deadlock freedom with a single VC. Overall, SEEC[3] reduces the router area by 73% over escape VC and ∼70% over SPIN and SWAP. DRAIN [28] has similar area and power overhead as SEEC as it

[3]mSEEC does not add any additional router complexity over SEEC as it only changes the route followed by the seekers.

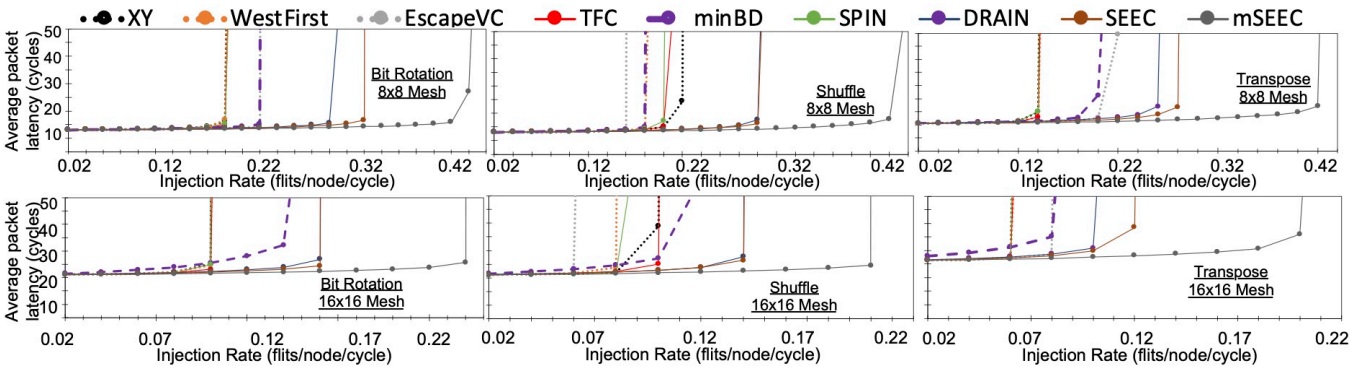

Figure 8: ***Latency versus injection rate for different traffic pattern across network sizes. SEEC and mSEEC outperform prior art.***

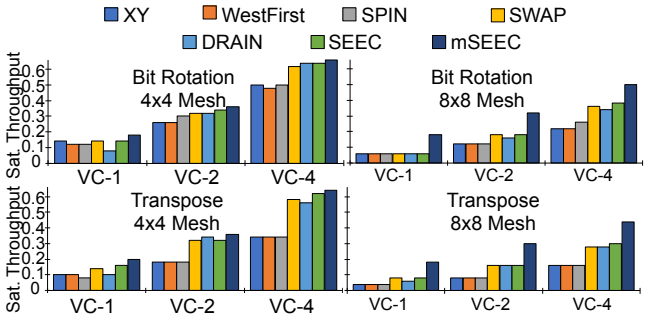

Figure 9: ***Saturation Throughput for Bit Rotation and Transpose traffic. Topology of increased size 4×4/8×8 Mesh.***

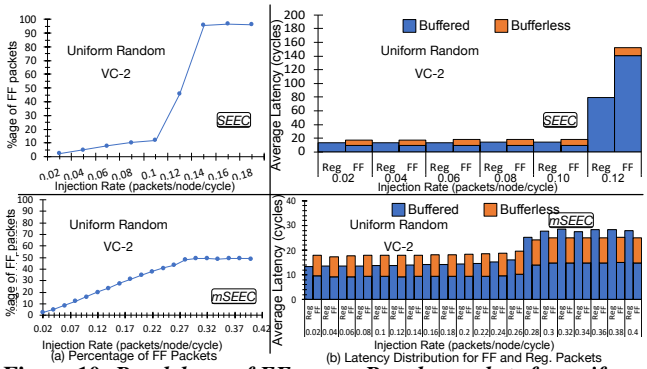

Figure 10: ***Breakdown of FF versus Regular packets for uniform random traffic on a 8x8 Mesh for SEEC and mSEEC.***

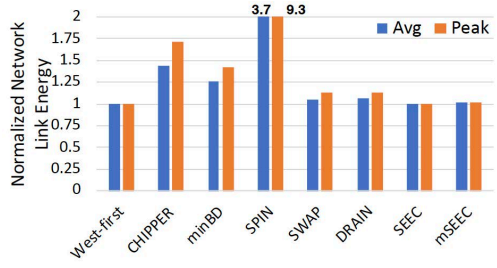

Figure 11: ***Average and Peak Network Link Energy across various deadlock-freedom schemes for uniform random traffic (VC=1).***

is also routing and protocol deadlock free with 1 VNet. DRAIN's overhead is the time-out counters and FSM for coordinating periodic drains [28].

4.3 Analysis with Synthetic Traffic

Fig. 8 shows latency-throughput curves for SEEC/mSEEC when compared against the baselines. Results are collected with a 4-VC per input port router configuration. SEEC is as good as or better than all prior works across different traffic patterns and topology sizes. On average across topology sizes, SEEC provides 65% higher throughput over the best proactive deadlock solutions (Escape VC[15] and TFC [21][4]), 50% over reactive (SPIN[35]) and 10% over other subactive solutions (SWAP[27], DRAIN[28]). Escape VC has lower throughput compared to SEEC as fully-adaptive minimal routing is restricted in 1 VC while SEEC allows fully adaptive minimal routing in all VCs. We discuss this in more detail in §4.4. SEEC provides 2×-3× throughput improvement compared to the MinBD bufferless scheme. mSEEC pushes the envelope further, due to multiple FF packets creating express paths in the NoC. Its relative performance over SEEC increases as topology size increases because of higher path diversity. mSEEC performs 20% better in 4×4, 25% better in 8×8 and 40% better in 16×16 Mesh on average than SEEC.

Saturation Throughput. Fig. 9 compares the throughput of SEEC/mSEEC with other schemes, across topology sizes and number of VCs per input port. Here SPIN, SWAP, SEEC, and mSEEC use fully adaptive random routing. mSEEC provides the highest throughput in all cases, followed by SEEC and then SWAP and DRAIN. A generic trend shown by these graphs is the decrease in saturation throughput with increase in topology size as expected. XY and WF routing suffer from the lowest throughput because they restrict the path packets can take in the NoC. SPIN suffers lower saturation throughput than SWAP, DRAIN, and SEEC because at saturation the deadlock detection algorithm kicks in and its probes hinder the forward movement of packets. SWAP and DRAIN have lower saturation throughput than SEEC because of misrouting. The difference between DRAIN and SWAP is that SWAP causes local pair-wise movement of packets while DRAIN proposes network-wide movement of packets. SEEC/mSEEC is free from such challenges and always routes packets minimally, while providing full path diversity.

[4]Note: Unlike the original paper [21], TFC does not show low-load latency improvement. Our baseline router is an optimized 1-cycle router, while the TFC paper's baseline was a 4-cycle router.

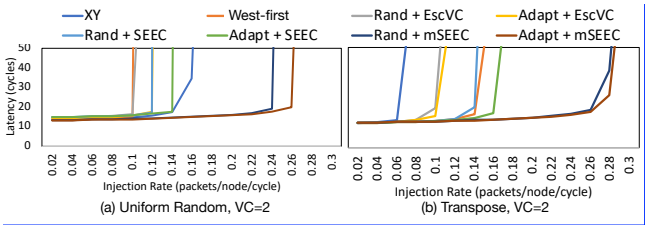

Figure 12: ***Comparison of Routing Algorithms: deterministic XY, west-first, Oblivious Random (Rand) and Adaptive Random (Adapt) across deadlock-free NoCs with 2VCs.***

Distribution of FF vs Regular Packets. Fig. 10(a) plots the percentage of FF packets (i.e., packets promoted to FF anytime during their traversal) received as a function of injection rate for uniform random traffic. It is not surprising to see that post saturation, heavy congestion and high likelihood of deadlocks in uniform random traffic [35] leads to almost every packet using SEEC to exit the network. For mSEEC, this number saturates to about 50%. For traffic patterns that have fewer average hops (e.g., shuffle) or are less deadlock-prone (e.g., transpose [35]), the percentage was found to saturate at ∼10-30%. Fig. 10(b) breaks down the latency distribution of the received packets (both regular and FF) across the buffered traversal versus the bufferless (i.e., FF) part of the traversal. A very interesting trend is observed here. While one may expect FF packets to be "faster" than regular packets, we observe the reverse, both at low injection and especially post saturation. This can be explained by the fact that the FF packets are those that were actually blocked at some router (and promoted via explicit seekers) and hence show a higher percentage of buffered time compared to unblocked packets that reached their destinations via a regular traversal. The bufferless component of the latency is quite small, as expected. These results point to potential future work on leveraging SEEC for QoS.

Energy Overhead Analysis. In Fig. 11, we plot the average and peak (i.e., at saturation) link energy of the NoC for uniform random traffic with a single VC. Results are normalized to West-first which has no misroutes. Notably, a reactive scheme like SPIN shows a huge spike in energy due to probes that are sent out upon timeouts to detect deadlocks and map their path. We see 3.7× on average and up to 9.7× at peak once deadlocks start to increase. Deflection schemes (minBD and CHIPPER) show 25-40% higher link energy on average and 45-74% at peak, making them efficient only at very low loads. Among subactive schemes, SWAP and DRAIN show about 5-6% higher link energy on average, which goes up to 12-14% near saturation due to increase in misroutes.[5] SEEC has no misroutes but does employ seekers to find packets. However, we find its energy overhead, both average and peak, is negligible, hovering at less than 1%, since it is a narrow sideband path with low activity compared to regular flits, as discussed in §3.6. In summary, SEEC provides significant performance enhancements over prior work with negligible energy overhead.

[5]We set the SWAP and DRAIN frequency to the default of 1024 cycles in their code [1, 2]. Reducing this showed ∼50% increase in peak link activity.

4.4 Discussion of Synthetic Traffic Results

Since SEEC and mSEEC provide both deadlock-freedom and congestion bypassing, a natural question is which of these two contribute most to performance improvement.

4.4.1 ***Lack of slowdown due to slower deadlock-resolution.*** A scheme like SEEC/mSEEC that lets cycles form and then subactively breaks them will lead to a drop in throughput when cycles start to form. However, we observed that this situation typically occurs after the network has already saturated and thus there is no visible slowdown due to subactive deadlock resolution. This is consistent with prior work that has argued for reactive [35] and sub-active [27, 28] deadlock resolution, over proactive that leads to performance loss, as we discuss next.

4.4.2 ***Impact of adaptive routing.*** To dig deeper into the performance benefits of SEEC, we ran a deep-dive experiment with 2 VCs, contrasting four routing algorithms: XY, West-first, Minimal Oblivious Random and Minimal Adaptive Random. XY is inherently deadlock-free but the other three are not. We created the following deadlock-free NoCs: (i) Both-VCs: XY (ii) Both-VCs: West-first (iii) VC0: oblivious random, (escape) VC-1: West-first (iv) VC0: adaptive random, (escape) VC-1: West-first, (v) Both-VCs: oblivious random with SEEC, (vi) Both-VCs: adaptive-random with SEEC, (vii) and (viii) same as (v) and (vi) with mSEEC. Fig. 12 plots the latency versus injection rate for two diverse traffic patterns.

For uniform random, XY routing is known to be highly effective and beats all routing algorithms except the ones with mSEEC. We also observe adaptive random providing higher throughput than oblivious random, for escape VC, SEEC and mSEEC. For Transpose, west-first has similar performance as oblivious random with SEEC, but adaptive random pushes throughput further. In both patterns, mSEEC provides the best throughput. The reasoning is as follows. SEEC and mSEEC not only support fully-adaptive minimal routing in all VCs, but also augment this with contention-free guaranteed FF-paths (enabled by seekers). At high-loads, *any* routing algorithm will lead to packet stalls when buffers fill up. But SEEC (and especially mSEEC) still allows links to get used in such situations via FF packets, increasing link utilization thus throughput. We elaborate on this effect next.

4.4.3 ***Impact of number of VCs.*** The throughput improvements of SEEC and mSEEC come from their ability to reduce Head-of-Line blocking, (despite having slow deadlock-resolution times as discussed above). This is because all baseline schemes (adaptive routing with turn-model, escape VC, SWAP, DRAIN, SPIN) are fundamentally still limited by finite number of VCs, which when full due to congestion, will lead to stalls. We demonstrate this in Fig. 12 where we contrast the performance of SEEC and mSEEC with 2 VCs against an escape VC router with increasing number of VCs for a set of synthetic traffic patterns. At 2 VCs, escape VC shows worse performance than both SEEC and mSEEC due to loss of path diversity within the escape VC. This is consistent with Fig. 12. As the escape VC gets more VCs, its performance improves. At 8+ VCs, we see escape VC's performance starts matching or beating SEEC and mSEEC. In contrast, in SEEC/mSEEC, the FF packets in SEEC/mSEEC can bypass full VCs, hopping over idle links all the way to the destination. *This effectively emulates the behavior*

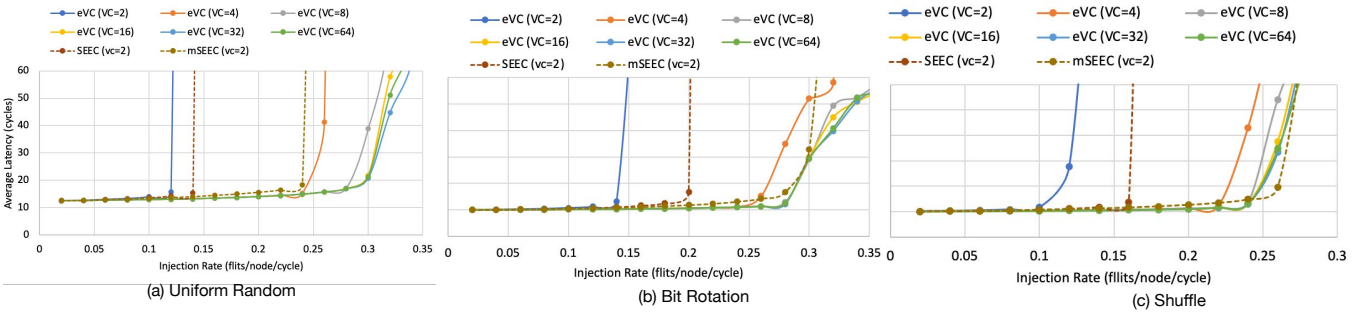

Figure 13: ***Performance of SEEC and mSEEC with 2 VCs, shown in dotted lines, against escape VC (eVC), shown in solid lines, with increasing number of VCs. The network is a 8x8 Mesh.***

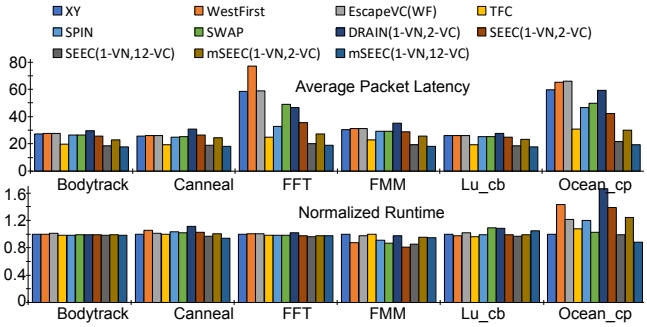

Figure 14: ***Average packet latency and normalized runtime (to XY routing) of applications in a 4×4 mesh.***

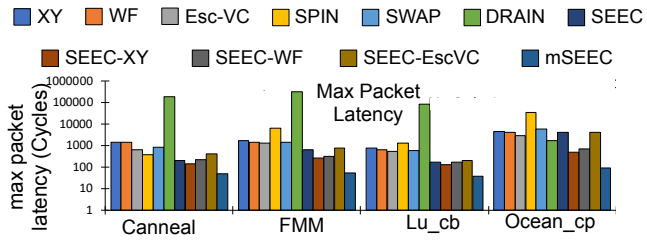

Figure 15: ***Max packet latency on*** 4×4 ***mesh (y-axis is log scale).***

of having more VCs without physically requiring more VCs in each router, enabling better performance at lower hardware cost.

4.5 Analysis with Application Traffic

Fig. 14 shows average packet latency and runtime improvement with SEEC/mSEEC when compared against the baselines. DRAIN and SEEC use one VNet, while the other schemes need 6 VNets to provide deadlock freedom (as dictated by the MOESI Hammer coherence protocol [7]). We evaluate SEEC/mSEEC with two configurations: (1) *iso-VC-VNet* and, (2) *iso-hardware* . In iso-VC-VNet, the number of VCs per VNet is a constant (2 VCs per VBet) irrespective of number of VNets; in iso-hardware, the total number of VCs per input port is same (i.e., 12 VCs with 1 VNet). The iso-VC-VNet SEEC (1-VN, 2-VC) performs similar to SPIN, slightly worse than TFC and better than XY, WF, Escape VC and SWAP on average packet latency at $1/6^{th}$ buffer area. At iso-hardware cost, mSEEC provides 40% improvement on average over all prior schemes. Both SEEC and mSEEC provide 5% average improvement in the total runtime of applications.

Impact on Application Tail Latency. Fig. 15 shows the tail latency incurred by packets in the network. On average, XY, West-first and Escape VC have similar tail latencies; however, SPIN has an order of magnitude higher maximum packet latency. This is because the expensive deadlock detection of SPIN prioritizes the movement of deadlock detection probes over actual packets [35]. This can further slow down the movement of actual packets in the NoC. In the extreme case, this leads to slowdown for certain packets. DRAIN has the highest tail packet latency among all other schemes due to frequent periodic misrouting of packets. SWAP performs similar to XY, WF and Escape VC. SEEC outperforms all baselines. SEEC has similar performance as SEEC-EscVC since for real traffic, fully adaptive routing within both versus a single VC does not buy affect the tail latency too much. However, tail latency results improve further when SEEC is augmented with XY routing. We observed an order of magnitude lower latency with SEEC-XY compared to other schemes. This shows that for these applications, fully-adaptive routing ended up increasing the tail latencies compared to XY. Further, enhancing XY with SEEC allowed it to get lower tail latencies than pure XY. This analysis shows that SEEC also benefits routing algorithms that are inherently deadlock-free.

5 CONCLUSION

We propose SEEC which sends special tokens called seekers from destination NICs to search for one or more blocked packets in the NoC and creates guaranteed express paths for them all the way to their destination. SEEC is the first work to provide both routing and protocol-level deadlock freedom and enhance network throughput via a unified solution. Moreover, SEEC can operate with a single VC in the entire NoC and does not add any turn restrictions, extra VCs, virtual networks, deadlock detection, or misrouting unlike prior art. In this work we demonstrate SEEC over non-faulty meshes. Extending SEEC to more complex systems with several clock domains and fault-tolerance is part of future work.

ACKNOWLEDGEMENTS

This work was supported by a Canada Research Chair and the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] .. DRAIN: Deadlock Removal for Arbitrary Irregular Networks. https://github.com/noc-deadlock/drain

[2] .. SWAP: Synchronized Weaving of Adjacent Packets for Network Deadlock Prevention. https://github.com/noc-deadlock/swap

[3] N. Agarwal, T. Krishna, L. Peh, and N. K. Jha. 2009. GARNET: A Detailed On-chip Network Model inside a Full-system Simulator. In *ISPASS*.

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

[5] Paul Baran. 1964. On distributed communications networks. *IEEE transactions on Communications Systems* 12, 1 (1964), 1–9.

[6] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques* (Toronto, Ontario, Canada) *(PACT '08)*. ACM, New York, NY, USA, 72–81.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[8] Xiao Canwen, Zhang Minxuan, Dou Yong, and Zhao Zhitong. 2008. Dimensional Bubble Flow Control and Fully Adaptive Routing in the 2-D Mesh Network on Chip. In *EUC*. 353–358.

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

[10] L. Chen, R. Wang, and T. M. Pinkston. 2011. Critical Bubble Scheme: An Efficient Implementation of Globally Aware Network Flow Control. In *IPDPS*. 592–603.

[11] 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.

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

[13] William J. Dally and Brian Towles. 2001. Route Packets, Not Wires: On-chip Inteconnection Networks. In *DAC*.

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

[15] 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.

[16] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko Lipasti. 2008. Circuit-Switched Coherence. In *International Symposium on Networks-on-Chip*.

[17] C. Fallin, C. Craik, and O. Mutlu. 2011. CHIPPER: A Low-complexity Bufferless Deflection Router. In *HPCA*. 144–155.

[18] Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu. 2012. MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect. In *2012 Sixth IEEE/ACM NOCS*. 1–10. https://doi.org/10.1109/NOCS.2012.8

[19] H. Farrokhbakht, H. Kao, K. Hasan, P. Gratz, T. Krishna, J. San Miguel, and N. Enright Jerger. 2021. Pitstop: Enabling a Virtual Network Free Network on Chip. In *Proceedings of the International Symposium on High Performance Computer Architecture*.

[20] Marina Garcia, Enrique Vallejo, Ramon Beivide, Miguel Odriozola, Cristobal Camarero, Mateo Valero, German Rodriguez, Jesus Labarta, and Cyriel Minkenberg. 2012. On-the-Fly Adaptive Routing in High-Radix Hierarchical Networks. In *Proceedings of the 2012 41st International Conference on Parallel Processing (ICPP '12)*. IEEE Computer Society, Washington, DC, USA, 279–288. https://doi.org/10.1109/ICPP.2012.46

[21] Amit Kumar, Li-Shiuan Peh, and Niraj K Jha. 2008. Token flow control. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 342–353.

[22] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K Jha. 2007. Express virtual channels: towards the ideal interconnection fabric. In *ISCA*.

[23] Hyoukjun Kwon and Tushar Krishna. 2017. OpenSMART: Single-Cycle Multi-hop NoC Generator in BSV and Chisel. In *Proc of the IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE.

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

[25] Vijay Nagarajan, Daniel J Sorin, Mark D Hill, and David A Wood. 2020. A Primer on Memory Consistency and Cache Coherence. *Synthesis Lectures on Computer Architecture* 15, 1 (2020), 1–294.

[26] Mayank Parasar. 2020. *Subactive Techniques For Guaranteeing Routing And Protocol Deadlock Freedom In Interconnection Networks*. Ph.D. Dissertation. Georgia Institute of Technology.

[27] Mayank Parasar, Natalie Enright Jerger, Paul V. Gratz, Joshua San Miguel, and Tushar Krishna. 2019. SWAP: Synchronized Weaving of Adjacent Packets for Network Deadlock Resolution. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. ACM, New York, NY, USA, 873–885. https://doi.org/10.1145/3352460.3358255

[28] Mayank Parasar, Hossein Farrokhbakht, Natalie Enright Jerger, Paul V Gratz, Tushar Krishna, and Joshua San Miguel. 2020. DRAIN: Deadlock Removal for Arbitrary Irregular Networks. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 447–460.

[29] Mayank Parasar and Tushar Krishna. 2017. Lightweight Emulation of Virtual Channels using Swaps. In *Proceedings of the 10th International Workshop on Network on Chip Architectures*. 1–6.

[30] Mayank Parasar and Tushar Krishna. 2019. Bindu: Deadlock-freedom with one bubble in the network. In *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*. 1–8.

[31] Mayank Parasar, Ankit Sinha, and Tushar Krishna. 2018. Brownian bubble router: Enabling deadlock freedom via guaranteed forward progress. In *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 1–8.

[32] Timothy Mark Pinkston. 1999. Flexible and efficient routing based on progressive deadlock recovery. *IEEE Trans. Comput.* 48, 7 (1999), 649–669.

[33] Timothy Mark Pinkston and Sugath Warnakulasuriya. 1997. On deadlocks in interconnection networks. In *Proceedings of the 24th annual international symposium on Computer architecture*. 38–49.

[34] V. Puente, C. Izu, R. Beivide, J.A. Gregorio, F. Vallejo, and J.M. Prellezo. 2001. The Adaptive Bubble Router. *J. Parallel Distrib. Comput.* 61, 9 (Sept. 2001).

[35] Aniruddh Ramrakhyani, Paul V Gratz, and Tushar Krishna. 2018. Synchronized progress in interconnection networks (SPIN): A new theory for deadlock freedom. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 699–711.

[36] Aniruddh Ramrakhyani and Tushar Krishna. 2017. Static bubble: A framework for deadlock-free irregular on-chip topologies. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 253–264.

[37] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W-K. Su. 1988. The Architecture and Programming of the Ametek Series 2010 Multicomputer. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications: Architecture, Software, Computer Systems, and General Issues - Volume 1* (Pasadena, California, USA) *(C3P)*. ACM, New York, NY, USA, 33–37. https://doi.org/10.1145/62297.62302

[38] Arjun Singh. 2005. *Load-balanced routing in interconnection networks*. Ph.D. Dissertation. Stanford University.

[39] Yong Ho Song and Timothy Mark Pinkston. 2003. A progressive approach to handling message-dependent deadlock in parallel computer systems. *IEEE Transactions on Parallel and Distributed Systems* 14, 3 (2003), 259–275.

[40] R. Wang, L. Chen, and T. Pinkston. 2013. Bubble Coloring: Avoiding Routing- and Protocol-induced Deadlocks with Minimal Virtual Channel Requirement. In *ICS '13*.

[41] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. *ACM SIGARCH computer architecture news* 23, 2 (1995), 24–36.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Update: ——– Each github repository, associated to the Zenodo DOIs, is now available for AE committee members for evaluation/inspection. Each github repo for (EscapeVC, SPIN, SWAP, DRAIN, and SEEC/mSEEC) will contain a script by name: ae_sc2021.sh (or ae_sc2021.py) at top level of the repo This will be used to generate the results present in _Figure 8_ of the paper. Evaluators need to run it using ./ae_sc2021.sh (or python ae_sc2021.py) After running the script Evaluators will observe "average_packet_latency" data which is used to create the graph for both 8x8 Mesh and 16x16 Mesh for Bit Rotation, Shuffle and Transpose traffic patterns. ——–

1. SEEC

SEEC is a technique to improve performance (reduced average packet latency) with deadlock freedom guarantee in NoCs.

2. Dependencies

SEEC is implemented in gem5 simulator, it modifies the on-chip simulator: Garnet present inside gem5 simulator.

Dependencies to compile and run the code are as follows:

Ubuntu 16.04 LTS:

sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python-dev python

Git: sudo apt-get install git

gcc 4.8+: sudo apt-get install build-essential

Install scons: sudo apt-get install scons

Python 2.7+: sudo apt-get install python-dev

Protobuf 2.1+: sudo apt-get install libprotobuf-dev python-protobuf protobuf-compiler libgoogle-perftools-dev

M4 macro processor not installed: sudo apt-get install automake

3. Get Code

Download from the following link:

https://www.dropbox.com/sh/uyjuy9mybq8wagu/AAD9YRKEw-nC3XtP73xglTlNa?dl=0

SEEC: ${gem5_seec}

DRAIN: ${gem5_drain}

SWAP: ${gem5_swap}

SPIN: ${gem5_spin}

EscapeVC: ${gem5_esc}

4. Compile the code:

scons ${gem5_*}/build/Garnet_standalone/gem5.opt -j9

5. Reproduce/Validate results

Build Synthetic: $gem5_seec/scons build/Garnet_standalone/gem5.opt -j9

Run Synthetic:

Baseline:

${gem5_seec}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –topology=Mesh_XY –num-cpus=64 –num-dirs=64 –mesh-rows=8 –network=garnet2.0 –router-latency=1 –sim-cycles=$cycles –inj-vnet=0 –vcs-per-vnet=${vc_} –seec=0 –one-pkt-bufferless=0 –num-bufferless-pkt=0 –injectionrate=${k} –synthetic=${bench[$b]} –routing-algorithm=${routAlgo} &

SEEC:

${gem5_seec}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –topology=Mesh_XY –num-cpus=64 –num-dirs=64 –mesh-rows=8 –network=garnet2.0 –router-latency=1 –sim-cycles=$cycles –inj-vnet=0 –vcs-per-vnet=${vc_} –seec=1 –one-pkt-bufferless=1 –num-bufferless-pkt=1 –bufferless-router=1 –injectionrate=${k} –synthetic=${bench[$b]} –routing-algorithm=${routAlgo} &

mSEEC:

${gem5_seec}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –topology=Mesh_XY –num-cpus=64 –num-dirs=64 –mesh-rows=8 –network=garnet2.0 –router-latency=1 –sim-cycles=$cycles –inj-vnet=0 –vcs-per-vnet=${vc_} –seec=1 –one-pkt-bufferless=1 –num-bufferless-pkt=1 –bufferless-router=8 –injectionrate=${k} –synthetic=${bench[$b]} –routing-algorithm=${routAlgo} &

SWAP:

${gem5_swap}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –network=garnet2.0 –num-cpus=64 –num-dirs=64 –topology=Mesh_XY –mesh-rows=8 –interswap=1 –whenToSwap=1 –whichToSwap=1 –sim-cycles=100000 –injectionrate=${k} –vcs-per-vnet=${vc_} –no-is-swap=1 –occupancy-swap=0 –inj-vnet=0 –synthetic=${bench[$b]} –routing-algorithm=${r}

SPIN:

${gem5_spin}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –topology=Mesh_XY –num-cpus=64 –num-dirs=64 –mesh-rows=8 –network=garnet2.0 –sim-cycles=$cycles –vcs-per-vnet=${vc_} –inj-vnet=0 –injectionrate=${k} –synthetic=${bench[$b]} –enable-spin-scheme=1 –dd-thresh=1024 –routing-algorithm=table –max-turn-capacity=40 –enable-variable-dd=0 –enable-rotating-priority=1

DRAIN:

${gem5_drain}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –topology=irregularMesh_XY –num-cpus=64 –num-dirs=64 –mesh-rows=8 –network=garnet2.0 –router-latency=1 –sim-cycle=$cycles –spin=1 –conf-file=${conf} –spin-freq=1024 –spin-mult=1 –uTurn-crossbar=1 –inj-vnet=${vnet} –vcs-per-vnet=${vc_} –injectionrate=${k} –synthetic=${bench[$b]} –routing-algorithm=0

EscapeVC (uses WestFirst Routing Algorithm in Escape VC, Normal VC use Adaptive Random Routing):

${gem5_esc}/build/Garnet_standalone/gem5.opt configs/example/garnet_synth_traffic.py –topology=Mesh_XY –num-cpus=64 –num-dirs=64 –mesh-rows=8 –network=garnet2.0 –router-latency=1 –sim-cycles=$cycles –inj-vnet=${vnet} –vcs-per-vnet=${vc_} –injectionrate=${k} –synthetic=${bench[$b]} –routing-algorithm=WEST_FIRST_

Full System:

SEEC (4x4 Mesh):

${gem5_seec}/build/X86_MOESI_hammer/gem5.opt –remote-gdb-port=0 ./configs/my_configs/boot_from_checkpoint.py –script=/usr/scratch/mayank/benchs/${benchmarks[$b]} -n 16 –topology=Mesh_XY –num_dirs=16 –mesh-rows=4 –network=garnet2.0 –vcs-per-vnet=${vc_} –routing-algorithm=${rout_} –seec=1 –one-pkt-bufferless=1 –num-bufferless-pkt=1 –bufferless-router=1 –inj-single-vnet=${single_vnet} –ruby &

mSEEC (4x4 Mesh):

${gem5_seec}/build/X86_MOESI_hammer/gem5.opt –remote-gdb-port=0 ./configs/my_configs/boot_from_checkpoint.py –script=/usr/scratch/mayank/benchs/${benchmarks[$b]} -n 16 –topology=Mesh_XY –num_dirs=16 –mesh-rows=4 –network=garnet2.0 –vcs-per-vnet=${vc_} –routing-algorithm=${rout_} –seec=1 –one-pkt-bufferless=1 –num-bufferless-pkt=1 –bufferless-router=4 –inj-single-vnet=${single_vnet} –ruby &

Author-Created or Modified Artifacts:

```
Persistent ID: DOI: 10.5281/zenodo.5171429
Artifact name: artifacteval\_sc2021

Persistent ID: https://github.com/noc-deadlock/seec
Artifact name: seec

Persistent ID: https://github.com/noc-deadlock/drain
Artifact name: drain

Persistent ID: https://github.com/noc-deadlock/swap
Artifact name: swap

Persistent ID: https://github.com/noc-deadlock/spin
Artifact name: spin

Persistent ID:
↪ https://github.com/noc-deadlock/escape\_vc
Artifact name: escape\_vc

Persistent ID: https://www.dropbox.com/sh/uyjuy9mybq⌋
↪ 8wagu/AAD9YRKEw-nC3XtP73xglTlNa?dl=0
Artifact name: dropbox-backup link
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: x86, quad core CPU with 8 or more GB RAM

Operating systems and versions: Ubuntu 16.04 LTS

Compilers and versions: gcc 4.8+, Python 2.7+, Protobuf 2.1+

Applications and versions: M4 macro processor, git, scons