AN ABSTRACT OF THE THESIS OF

Ryan Gambord for the degree of Master of Science in Electrical Engineering presented on November 23, 2021.

Title: Network Abnormalities in Routerless Networks on Chip

Abstract approved: _____

Lizhong Chen

Routerless networks on chip (NoCs) were recently introduced as an alternative to traditional mesh-based scalable networks, promising improved performance and scalability with a large decrease in power and area usage by the elimination of routing on in-flight packets. Without mitigation, most network designs, including routerless, are susceptible to loss of function or total failure through livelock, starvation, and deadlocks under certain conditions. In this work, I demonstrate the inadequacy of existing proposed mitigation schemes and propose revised schemes through careful theoretical treatment of the problem of network abnormalities. This revised design requires fewer physical resources than the existing design, yet outperforms it under simulated benchmarks. The revised network design achieves higher throughput and lower latency at all injection rates, and can sustain higher injection rates before saturation.

Network Abnormalities in Routerless Networks on Chip

by

Ryan Gambord

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 23, 2021
Commencement June 2022

Master of Science thesis of Ryan Gambord presented on November 23, 2021.

APPROVED:

Major Professor, representing Electrical Engineering

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Ryan Gambord, Author

ACKNOWLEDGEMENTS

The author expresses sincere appreciation to professor Lizhong Chen for his guidance and input on this research effort.

In memory of Bear, a true friend and attentive listener.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

1.1 The State of Networks On Chip

Dual core processors, first introduced in the early 2000s, saw multiple independent processing units seamlessly sharing a single memory/io resource through an on-chip interconnect, over which both data and control messages would be carried. Simple multiplexed buses were already in use at the time to connect the several components of a processor together [10], and the addition of a second core came with few serious conceptual changes to this design beyond the addition of an interconnect interface that implemented relatively simple cache coherence protocols to avoid data hazards and enforce sequential consistency across memory accesses between separate processing units. These early NoCs serviced a few heavyweight cores and generally were blessed with plentiful resources and low performance requirements. Scalability for such small distances and few nodes was not a primary concern, and early networks tended to consist of simple bus, tree, or crossbar type indirect networks, where all nodes surrounded a central network switching circuit with endpoints around its perimeter. As needs grew and resources disappeared, these earlier designs became strained by their inefficient use of resources at scale.

Inherent scaling limitations of simple buses were readily apparent as architects sought to increase core counts [3]. On a theoretical level, a bus must allocate a fixed

bandwidth among all connected devices, decreasing performance proportionally to the number of connected devices. This performance can be recovered only by increasing the clock rate of the bus or widening it. Practically, however, widening a bus beyond the largest data transmission brings no improvement, and increasing the clock rate increases power usage with the cube of its frequency. Eventually, physical power dissipation thresholds and signal propagation delays along the physical bus place a hard limit on the clock rate of a bus for a given length, effectively constraining the scalability of a bus-based interconnect. Indirect networks suffered similar problems and their centralized nature made scaling particularly untenable.

For these reasons, buses and indirect networks have largely fallen to the wayside in favor of direct networks, in which packets hop between routers at each node they visit along the path towards their destinations. These routers have long provided a good balance between performance and scalability while resources for their implementation have remained abundant. Their tiling design characteristic allows for the addition of nodes at a fixed cost, generating a linear growth characteristic. Performance in direct networks drops off smoothly with distance between nodes because physically adjacent nodes tend to have a correspondingly close topological distance in a direct network.

The first major iteration on interconnect design came about as the unavoidable segmentation of large buses into smaller isolated units separated by buffers to overcome the scaling limitations on throughput and clock rate. Spatial localization of data meant that neighboring elements could now communicate independently from the rest of the network, dramatically increasing throughput for traffic within a neighborhood, but with a corresponding increase in latency between distant elements. These bus-

and-buffer units could be connected in an endless series of repeating tiles, forming the first truly scalable ring networks.

Rings are lightweight and performant, often outperforming the components they connect, but problematic latency scaling characteristics motivated further development among early many-core processors. As before, the ring network was broken up into smaller sub-rings, with on-network routers mediating the connections between sub-rings. A canonical realization of this design, the mesh network, consists of nodes arranged in a grid with bi-directional half rings along each row and column. Typically, a dimension-ordered routing scheme is implemented so that packets always move along one dimension, and then the other, to reach their destination.

Router-based networks have several major performance advantages over ring networks. Latency scales with the nth root of the number of dimensions of the network/routers; with 2 and 3 dimensional topologies being the most common [13]. The design space for routers is also much larger than for rings, making routers a popular topic of research interest. First developed with experimental many-core networks in mind, mesh networks are increasingly appearing in multi-core chips, such as Intel's Mesh Interconnect Architecture, implemented in several Skylake processors, chipping away at the dominance of ring networks.

1.2 Routers: Too Big to Fail?

1.2.1 Adapting to a Changing Philosophical Landscape

In recent years, researchers have begun to abandon heavyweight processing cores and super-scalar pipelines in favor of a minimalist design philosophy that favors a large number of lightweight processing cores, expected to reach into the thousands, shortly [6, 7]. These many-core processors will implement reduced instruction sets to efficiently execute parallel workloads within tight area and power constraints [8]. In contrast, conventional router-based NoCs can easily overshadow these cores with a significant overhead in die area and power usage [11,12,17]. Furthermore, conventional network performance may not be sufficient to serve such a large number of nodes. The three major components of a router are a pool of buffers, the logic to implement the routing algorithm, and an internal crossbar to connect all of the links together. A variety of research efforts have targeted these three components for improvement.

Conventional routers implement a store-and-forward policy, requiring at least one packet-sized buffer for each virtual network, the number of which is determined by the cache coherence protocol. A router may contain several times the minimum number of buffers to increase network elasticity and reduce the effects of congestion. One method of reducing buffer overheads is to implement cut-through traversal of the network; rather than queuing entire packets at each hop, a packet is allocated an entire multi-hop route and traverses the route directly. With this strategy a router only need single-flit, rather than packet, sized buffers, since an entire packet will be spread across as many nodes as it is long during transmission. Another major

advantage is significantly reduced latency, proportional to the length of the packet, due to the absence of full queuing at each hop. Another method of reducing the need for buffers is to implement hot-potato routing. A router always has as many outgoing as incoming links, so it is possible to route every incoming packet out of the router immediately, just not in the packets' preferred direction (misrouting).

At the core of each router is a large crossbar connecting all of the incoming and outgoing links together. These crossbars use a considerable amount of area and power, and scale quadratically with link width or number, limiting the ability of a router-based network to improve throughput. Reducing the dimensionality of these crossbars is critical, and a simple technique to do this is through turn-restriction – disallowing routing from a particular incoming link to a particular outgoing link. In theory, and often in practice, all turns that don't disconnect the network graph can be removed. For example, Intel's Mesh Interconnect Architecture implements X-Y routing, where packets travel in one direction only along the vertical axis, then turning only once to travel along the horizontal axis to reach their destination. Another popular technique is to increase crossbar frequency but halve its width by interleaving each flit across two cycles, a technique called double-pumping [11, 18].

1.2.2 Missing the Mark on Manufacturing

Conventional router-based networks are topologically constrained. Router architecture is intrinsically linked to the overlying topology of the network – changing topology requires significant change to the router itself, which must be optimized and

validated for every particular design. Tessellating of routers practically precludes irregular topologies, because the logic for the routing algorithm, allocation of buffers, and structure of the internal crossbar would need to be redesigned for any local variation in topology, and then the entire network would also need to be validated in its specific configuration. While theoretically possible, these limitations are major constraints in the design space and speed of development of physically realizable networks on chip.

Routers are also hurtling towards a manufacturing landscape that they are ill-equipped to adapt to. We are only a few generations from reaching physical feature size limits and the end of Moore's law. Meanwhile, wiring resources are more abundant than ever as manufacturing processes and materials science continue to advance. Unfortunately, routers simply can't use these extra wires, because their internal crossbars scale quadratically with wire usage, and the routing algorithm's complexity grows factorially with each additional link. Each of these scaling behaviors represent large outlays of silicon in proportion to additional link capacity, a reversal of the trends in manufacturing capabilities.

This race to streamline and simplify routers to meet changing network demands signals the coming obsolescence of router-based NoC research, and the need for a new class of scalable, low-power next-generation of networks on chip.

1.3 Routerless Networks on Chip

Each of the aforementioned interventions is reductionist in nature, sacrificing some level of functionality or quality of service guarantee in favor of simplifying the network. Unifying these schemes – fully restricted turns, bufferless hot-potato routing, and predefined paths through the network – results in a completely novel class of network, consisting of a number of overlapping ring subnetworks. This type of network is most appropriately described as a routerless network, a name first proposed in [2]. The complexity of the routerless interface also scales (sub-)linearly with increasing wire utilization, avoiding the manufacturing pitfalls plaguing routers. In essence, routerless networks exist on a design-space continuum between ring- and router-based networks; routerless may be equally imagined as an extremely restricted router architecture, or as a natural extension of ring architecture, and routerless blends characteristics of both to produce a network with a balance of simplicity, scalability and performance. However, routerless is not simply a re-imaging of existing networks – it's a statement about the future of network design.

As logic resources become scarcer, interfaces will become constrained; meanwhile, abundant wiring resources will push topology into the spotlight as the central aspect of network design. Routerless is a step in this direction; the primary purpose of [2] was to introduce a functional and efficient routerless interface as a proof-of-concept, which built off of a similar prior work that focused on topology of multi-ring networks [15]. The result was almost an order of magnitude reduction in area and power and nearly doubled performance metrics compared to a conventional mesh network.

Shortly after [2] was published, a follow-up paper, [14], demonstrated the feasibility of deep-reinforcement learning as a tool to explore the large topological design space of routerless and optimize network layouts to meet different constraints, illustrating the extreme flexibility of the routerless design for adoption in a wide variety of contexts.

1.3.1 Design Details

The routerless network topology is comprised of a number of overlapping loops, of various sizes and shapes, which operate much like independent ring networks to transport packets across the network. Packets are fixed to travel along whichever loop they are injected onto, so there is no need for an internal crossbar or routing logic, as every incoming link is hardwired to a particular outgoing link. Loops implement bufferless routing, so only a single-flit buffer is required at each hop to hold the currently transiting flit; on each cycle it accepts a new flit from upstream, and sends the old flit downstream. This represents the entirety of the on-network portion of the interface; the remainder of the routerless interface exists solely to mediate the connection between the node and the loops, i.e. injection and ejection of packets.

Injection and ejection are mediated by a simple multiplexer/demultiplexer system that joins all the loops on the interface to its externally facing ports. At the beginning of each cycle, the interface polls the incoming link buffers for candidates for ejection. Any available ejection ports are assigned to these candidates following a round-robin scheme to enforce fairness between the loops. Remaining packets are left alone to recirculate on their loops to return at a later time and reattempt ejection.

When a packet reaches the front of the injection queue, the interface looks up its destination in a simple table to determine which loop to inject the packet onto. It then waits until that particular loop's incoming flit buffer contains either a head flit or a bubble before injecting; this prevents injection of one packet inside of another (nesting). If flits arrive during the injection of a packet, contention for the outgoing link occurs; since a partially injected packet cannot be recalled, any incoming flits *must* be delayed until the injection process is complete. To facilitate this, the interface contains a pool of special extension buffers (EXB), which are simple FIFOs. During injection, an EXB is spliced into the loop between the incoming link and its buffer. Any arriving flits are diverted into the EXB until injection is complete. Following injection, the EXB continues to carry traffic as part of the loop. On cycles when the EXB does not receive a flit, its length shrinks until it is empty, at which point it is excised so that it can be re-used for the next injection of a packet. Importantly, the pool contains fewer EXBs than there are loops, and they are shared among the loops. In [2], it was found that more than two EXBs had only a very small performance improvement.

The topology of the routerless network is highly flexible to meet specific constraints. The only requirement is that every pair of nodes must be connected by at least one loop. As a result, it is not necessary for any particular loop to visit any particular node, and it is not necessary for any particular loop span the entire network. Once the network is fully connected, additional loops may be added to improve the performance characteristics of the network, but are not necessary for correctness.

1.4 Objectives

1.4.1 Introduction to Network Abnormalities

Network abnormalities are rare conditions which can occur as a result of interactions between dynamic properties of the network and specific patterns of traffic. Packets may be unable to inject, get stuck somewhere on the network, never reach their destinations, or be unable to eject when they do. Although these events occur rarely, their consequences can be catastrophic to the functioning of the system. Formally, these conditions are classified as either livelock, starvation, or deadlock.

Livelock occurs when a packet is able to continue moving on the network but is never ejected by its destination. This is a concern in any network that allows packets to travel in unproductive directions, such as is the case with bufferless/hot-potato networking. Starvation occurs when a node is prevented from ever injecting a packet due to its inability to allocate a necessary resource. This can occur if the desired outgoing link is consistently busy or blocked, and is an issue across a variety of network types whenever nodes don't naturally exert back-pressure to relieve themselves of congestion. Finally, deadlock occurs whenever a packet is prevented from ever moving, generally due to a cyclic resource dependency that becomes fully occupied. Deadlock susceptibility often depends on interactions between several network elements and can be difficult to recognize, but cut-through routing is a good example of a deadlock-prone scheme because the tails of packets can block the heads of others, forming a cycle of packets successively blocking one another.

1.4.2 Considerations For Theoretical Approach

Because these conditions occur rarely, and often under already adverse traffic conditions, feasible solutions focus either on structural changes to prevent abnormalities with minimal performance degradation under typical operation, or recovery schemes with minimal resource overheads (since these resources are rarely used in actual practice).

There are three general classes of solutions, known as ostrich algorithm, stochastic/probabilistic, and deterministic. The ostrich algorithm is a popular solution that consists of simply ignoring the problem entirely, assuming that the likelihood and consequence of failure is acceptably low for a given application (often, the consequence of failure is rebooting a personal computing device). Next is stochastic or probabilistic solutions, which guarantee abnormalities will eventually spontaneously resolve given enough time under an assumption that traffic patterns are random to some degree, and is an acceptable trade-off between reliability and cost for most purposes. Finally, deterministic solutions are guaranteed to resolve abnormalities within some bounded time interval regardless of traffic pattern, and are necessary for critical hardware that cannot accept failure under any condition.

1.4.3 Outline

The objective of this work is to explore the problem of network abnormalities in routerless networks. In this work, I will develop a formal theoretical treatment of network abnormalities in a general sense, which will then be applied to evaluate the

only other existing design, proposed in [1], for correctness. After demonstrating the fallaciousness of the existing schemes, I will propose revised deterministic schemes that are provably correct solutions to network abnormalities in routerless networks, and comparatively evaluate the performance characteristics of both designs using both synthetic and full-system benchmarks.

Chapter 2: Formal Theoretical Analysis of Network Abnormalities

The major design goals of any scheme that addresses these issues arise from their rarity and severity. As extremely rare events, avoidance schemes should use a minimal amount of resources in terms of overhead, and, most importantly, have minimal impact on normal operation. Due to their severity, correctness under strict constraints must also be guaranteed through formal verification.

Verification of network abnormalities first requires formalization of the problem to be solved. Each of livelock, starvation, and deadlock can be formalized as follows:

> **Livelock** $\rightarrow$ A packet exists which does not eject in a bounded interval.
>
> **Starvation** $\rightarrow$ A necessary resource does not regenerate in a bounded interval.
>
> **Deadlock** $\rightarrow$ A cyclic release-after-reservation resource dependency exists *AND* every resource in the cycle may be simultaneously occupied

2.1 Livelock

Livelock is a condition in which a packet continues moving through the network, but is never ejected. In routerless, when a packet reaches its destination but cannot eject, it is left to recirculate on the loop. If the the ejection port is always busy when the packet arrives, then it can become livelocked. This can trivially occur if a livelocked

packet is circulating and every time it arrives at its destination, the destination is busy ejecting another packet that arrive on a previous cycle. Timing the injection of these earlier packets to always arrive right before the livelocked packet returns allows for indefinite circling of the packet.

Livelock is a potentially serious problem in routerless networks; it can occur at an extremely low injection rate, requiring a fresh packet to be injected only once per revolution of the livelocked packet. Under benchmark simulations, less than 1.6% of packets ever recirculated, and no packets were recorded to recirculate more than three times. However, any program that contains a loop that generates a packet at a specific rate and that has an exit condition that depends on delivery of a livelocked packet could fail to finish executing.

In [2] the authors propose a simple livelock scheme consisting of a one-byte lap counter that is incremented every time a packet fails to eject at its destination. When the lap counter reaches a threshold value of 254, the interface reserves the next available ejection port for when the packet next arrives.

While simple, this scheme has two major problems. The first major issue is that multiple packets may reach the livelock threshold value of 254 at the same time, in which case the reservation of the ejection port is undefined. Second, if the ejection port is blocked due to off-network conditions, then reservation of the ejection port before the livelocked packet returns might not be possible. This results in undefined behavior regarding the packet's lap counter. Allowing it to overflow would allow a packet to repeatedly become livelocked, fail to eject, and then return to a counter value of 0, indefinitely. Preventing it from overflowing maintains the packet's livelocked status,

but allows younger packets to catch up to it, potentially stealing its reservation when the ejection port becomes available, repeatedly. A more rigorous approach is necessary to ensure that these conditions do not arise.

Before developing a general livelock scheme, I first present two core constraints. First, loops are treated as black boxes of an indeterminate (but finite) length and number. Second, the ejection port is also treated as a black box that can be next reserved in an unknown and arbitrarily long (but finite) period of time. These two constraints preclude trivial schemes that simply apply some unique ID to packets, or tracks a raw hop or lap count, because neither the number of packets, nor the amount of time on the network are bounded; this would result in arbitrarily large ID or counter field sizes in packet headers.

Observing that allowing multiple packets to become simultaneously livelocked, and that this might result in undefined behavior in reserving the ejection port motivates us to propose that each loop should operate in two modes, normal operation, and livelock recovery mode. Under normal operation, the interface operates just as in the previous scheme; however, when a packet reaches the livelock threshold, the loop is placed in livelock recovery mode.

When a loop is in livelock recovery mode, the node stops incrementing packets' lap counters. This guarantees that the node never has more than one livelocked packet on each loop. Whenever the ejection port becomes available, it is allocated to livelocked loops in a round-robin fashion. Once a livelocked loop ejects its livelocked packet, it returns to normal operation.

This scheme requires the addition of a simple round-robin counter and bitfield to

track which loops are in livelock recovery mode. It also requires only a minimum of 2 bits in the lap counter to function, rather than the 8 bits proposed in the original design.

Lemma 1. *The revised livelock scheme described above is livelock-free.*

Proof. (by contradiction) Assume that the revised livelock scheme is not livelock-free.

This implies that there is a packet, P, that is never ejected.

Either,

1. P's lap counter reaches the livelock threshold value:

   The ejection port will eventually be reserved for the loop P is on and P will be ejected.

   Contradiction.

   OR

2. P's lap counter does not reach the livelock threshold value:

   (a) This implies P's lap counter reaches some upper bound that it does not cross.

   (b) (a) implies that P's loop is in livelock recovery mode whenever P arrives; otherwise P's lap counter would be incremented.

   (c) Livelock recovery mode implies that the loop has a packet that has reached the livelock threshold value. According to (1), such a packet must be eventually ejected.

(d) (b) and (c) implies an infinite supply of livelocked packets that are not P

(e) (d) implies that packets younger than P are becoming livelocked, while P is not.

(f) However, the order of packets on the loop never changes and the lap count increment circuit enabled and disabled at the same point in that ordering, so it is impossible for a younger packet to surpass an older packet in lap count.

Contradiction.

□

2.2 Starvation

Starvation is a condition in which an interface is unable to inject a packet because it lacks a necessary resource to do so. In routerless, when an EXB is used to inject a packet, it remains spliced into the loop until it receives enough bubbles to free itself. If the upstream node does not provide any bubbles, then the EXB can become stuck in the loop. If all of the EXBs in the interface become stuck in this way, then the interface is starved. The most trivial example of this is if the node immediately upstream from the starved node consumes every bubble it receives, and injects a packet in its place. This prevents the starved node from ever receiving any bubbles.

The scheme proposed in [2] requires the addition of a simple counter attached to each EXB that counts the number of cycles that the EXB is committed to a loop.

When the counter reaches a threshold value to indicate starvation, the EXB tags the next packet it sees with the starved node's address (with a special starvation address field in the packet header). When the packet is ejected at its actual destination, instead of being completely removed from the network, it is converted to single-flit ghost packets that are addressed to the starved node. When these ghost packets arrive at the starved node, they are converted directly into bubbles and used to drain the EXB. If this does not produce enough bubbles to drain the EXB, the process is repeated until it is empty.

Lemma 2. *The original starvation scheme described above is starvation-free.*

Proof. (by contradiction) Assume that the starvation scheme is not starvation-free.

This implies that there is a node, N, that never receives bubbles on a loop, L, that is has an EXB committed to. In other words, N sees an endless stream of packets (or ghost flits) destined for other nodes on L.

Either,

1. N is able to tag a packet: lemma 1 guarantees ejection of the packet, which generates ghost flits destined to N. At N the ghost flits are converted to bubbles.

Contradiction.

OR

2. N does not see any untagged packets (they are either already tagged, or ghosts):

I denote the list of starved nodes on L as $\mathcal{S} = [S_a, ..., S_N, ..., S_b]$, where $[S_a, ..., S_{N-1}]$ feed only tagged packets or ghosts to $[S_N, ..., S_b]$. This means that there are nodes upstream of S_a that are always injecting packets, but never receive bubbles, implying that they either become starved, or receive more bubbles or packets to tag from yet more upstream nodes. Essentially, bubbles are moving from upstream to downstream steadily, requiring an infinite amount of upstream nodes to prevent bubbles/packets to tag from ever reaching S_N. This this is not possible, S_N eventually tags a packet or receives a bubble. This repeats indefinitely until all of the nodes in $[S_N, ..., S_b]$ are freed, including N.

Contradiction.

□

While lemma 2 demonstrates that the original scheme is, in fact, starvation free, it can be improved upon. The proof of lemma 2 results from the net downstream flow of bubbles/taggable packets in the network. This can be enforced by simply ensuring that, when a starved node sees a packet, that the bubbles produced by that packet return to the packet's source node, and are emitted downstream of the source node. This ensures that whichever bubbles are used to inject a packet at the source node can only be used to inject a new packet downstream of the source of that packet. Through repeated iteration, this downstream injecting node must eventually be the starved node.

Formally, rather than a starved node address field, each packet has a single bit *starved* flag. When a node is starved, indicated by a packet waiting to inject for

longer than a set threshold, it sets this flag on every packet that it sees on any loop connected with an extension buffer attached to it. If the flag is already set (by an already-visited starved node), it remains set. On ejection of a packet with the *starved* flag set, the destination address of the packet is updated to take the same value as the packet's source address. Since no packets normally have a matching source and destination address, this identifies the packet as a special ghost packet. When the ghost packet arrives at the node immediately downstream of the node corresponding to its matching source and destination fields, it is converted into bubbles.

This scheme is an improvement over the original scheme because it requires only one bit in the packet header regardless of loop size; in contrast, the original scheme requires a full address field in the packet header, which grows with increasing network size. There is also no longer a need for a node to track whether it has an actively tagged packet, as with previous scheme, and there is no need to check that a packet hasn't already been tagged.

Lemma 3. *The revised starvation scheme described above is starvation-free.*

Proof. (by contradiction) Assume that the starvation scheme is not starvation-free.

This implies that there is a node, N, that never receives bubbles on a loop, L, that is has an EXB committed to. In other words, N sees an endless stream of packets (or ghost flits) destined for other nodes on L.

Either,

1. N is able to tag a packet: lemma 1 guarantees ejection of the packet, which generates ghost flits destined to N. At N the ghost flits are converted to

bubbles.

Contradiction.

OR

2. N does not see any untagged packets (they are either already tagged, or ghosts):

   There must be an infinite stream of packets in the network to starve N.

   If N_{-k} injects infinite packets, this results in bubbles produced somewhere between N_{-k} and N. These bubbles must be consumed before reaching N, but this requires a node between N_{-k} and N to inject infinite packets, in turn. Thus, any node that injects infinite packets must have a node between itself and N that also injects infinite packets; however, this results in infinite nodes, implying that there is no node which injects infinite packets.

   Contradiction.

□

2.3 Deadlock

Deadlock is a network abnormality that occurs when packets halt on the network for an indefinite length of time. This requires two conditions to be met:

1. A cyclic resource dependency exists on the network, where each resource must reserve the next resource before it can be freed.

2. All resources on the cycle are able to be simultaneously occupied.

There are two types of deadlock that can occur in a network, routing-dependent deadlock and message-dependent or protocol-dependent deadlock. Routing-dependent deadlock involves only the routers on the network. Generally, networks prevent routing-dependent deadlock by removing any cyclical paths for packets to travel along. In routerless, since packets travel on cyclical loops, routing-dependent deadlock is instead prevented by the bufferless routing strategy because all of the flits in the loop move simultaneously.

In routerless, the starvation scheme depends on the livelock scheme to reliably produce bubbles and free EXBs stuck in loops. However, if it is possible for a packet to become livelocked because its destination node is starved, this creates a cyclic dependency between the two schemes. This can occur when ejection of a request packet requires the injection of a corresponding reply packet. If the node is starved and unable to inject replies, it may be unable to then eject requests. If the loop is fully occupied with requests, all destined to starved nodes, deadlock occurs. This is referred to as protocol-dependent deadlock because the cache coherency protocol creates a dependency between injection and ejection that would not have otherwise existed.

[2] proposed a deadlock prevention scheme that consists of adding a special escape buffer to the interface. This escape buffer could be used to temporarily hold a request packet when the ejection port is blocked due to starvation and a backlog of reply packets to inject. The injection port then uses the space the request packet leaves behind to immediately inject a queued reply packet. This allows the ejection port to

drain one packet, so the temporarily held packed can then be moved from the escape buffer into the ejection port.

Careful analysis shows that this scheme is functionally equivalent to simply extending the ejection port to hold an extra packet in its queue, and therefore preserves the problematic cyclical resource dependency that allows deadlock to occur in the first place. To its benefit, the addition of resources in the cycle does require even more packets to queue before deadlock can occur, making the event significantly less likely. However, a major flaw in the design is found in the fact that it immediately injects a queued reply packet in the space left by the removed request packet; this assumes that the length of the request packet is at least as long as the length of the reply packet. Otherwise, the node requires an EXB to ensure there is no collision with subsequent incoming packets during injection.

A functional deadlock scheme must either break the cyclic dependency or prevent simultaneous occupation of all resources in the cycle. Prevention of simultaneous occupation of resources is similar to previous designs such as critical bubble, where a bubble is always maintained in the cycle to prevent deadlock. Such a scheme, however, would cause a tremendous performance impact on routerless networks. Such a bubble would sit in an unused EXB most of the time. A node experiencing protocol-level deadlock conditions would then need to send some sort of message to release this bubble onto a specific loop. However, if the node that is holding the bubble already has an EXB committed to that loop, it won't be able to fulfill the request. Attempting to tie up these sorts of loose ends results in a scheme with a very high performance and logic overhead, but is expected to be rarely, if ever, triggered. For this reason, I

instead look to breaking the cyclic dependency that exists.

I observed that deadlock cannot occur on the loop itself, because flits on a loop are always moving; instead, deadlock happens inside the interface where the ejection port depends on the injection port between request and reply packets. To resolve this issue, I propose that each interface has a separate ejection and injection port for each message class. In conjunction, each message class's injection port has a corresponding EXB that is used only for that message type. Unlike the previous design, in which general purpose EXBs were as long as the longest message, in this case each EXB is only as long as the message type that it corresponds to. Furthermore, I also ensure that bubbles originating in a reply message class's EXB cannot ever be used to inject a request packet; this is accomplished by having bubbles retain the message class designation of the packets they originate from. These class-designated bubbles can only be consumed by their corresponding EXBs. Overall, this serves to completely segregate message classes wherever they might come into conflict with each other.

Because no cyclic resource dependency exists in this case, the revised scheme is deadlock free, and no other verification is required.

Chapter 3: Material and Methods

The livelock and starvation schemes are expected to have no impact on normal performance because they only trigger under high loads, nor are they expected to significantly affect power or area due to their simplicity. However, the deadlock scheme requires significant changes to the structure of the routerless interface, and must be evaluated for performance effects under synthetic and full-system benchmarks.

As a significant component of this research effort, I implemented the routerless network with both baseline and revised schemes for synthetic [1] and full-system evaluations in Gem5 [16]. Gem5 is a cycle-accurate simulator that is capable of running a variety of full-system benchmarks on an unmodified Linux kernel. Along with implementing routerless in Gem5, I also patched and compiled PARSEC [4, 5] benchmarks to run inside the simulator for full-system stress testing of the routerless network. PARSEC is a suite of multi-threaded benchmarks designed to evaluate next-generation multi-core systems.

Synthetic		Full-System	
Bit Complement	Bit Reverse	Blackscholes	Bodytrack
Bit Rotation	Neighbor	Canneal	Dedup
Shuffle	Tornado	Ferret	Fluidanimate
Transpose	Uniform Random	Freqmine	Streamcluster
		Swaptions	Vips

Table 3.1: Listing of Evaluation Benchmarks

All tests were done on a 4x4 network for both the baseline and revised schemes. Three topologies were evaluated: Baseline, DRL-generated, and Balanced. The Balanced topology is a minor modification to the Baseline topology where the direction of certain loops is reversed so that nodes have equal numbers of incoming and outgoing links in each given direction. Eight synthetic and ten full-system benchmarks were used in evaluation, shown in Table 3.1. The synthetic benchmarks were run for 100,000 cycles each with injection rates from 0.001 to 1 in increments of 0.001 (per cpu, per cycle), and the full-system benchmarks were run to completion. The cache coherence protocol contained three virtual networks, two of which carry single flit packets, and the third which carries five flit packets. For the revised deadlock scheme, each vnet was allocated two EXBs per interface, for a total of 2x5 + 4x1 = 14 flit buffers. The baseline scheme was allocated three EXBs per interface, for a total of 15 flit buffers. This allowed comparison of performance with an expectation of comparable area and power requirements in the revised scheme (one fewer flit-sized buffer, offset by slightly more switching logic in the revised scheme).

Chapter 4: Results and Analysis

Global (on-network and queuing) latency, throughput, livelock recovery and starvation recovery statistics were gathered. Full-system benchmarks reported (simulated) time spent in the region of interest (ROI), which is correlated directly to cycle counts and throughput through the simulation clock rate. The Balanced and Baseline topologies showed nearly identical results, so the Balanced topology was discarded from analysis. The Baseline topology had a large amount of noise in synthetic benchmarks, and did not converge clearly for several benchmarks, unlike the DRL topology. For full-system, only DRL results were gathered due to time constraints, and with consideration of the quality of the data gathered previously with synthetic benchmarks guiding that decision. Figures show only DRL results as they most clearly show the difference in performance between the baseline and revised abnormality schemes, although the results for the other topologies were taken into account in overall analysis. I confirmed that the Baseline and Balanced topologies showed the same trends as the DRL topology, but that they were less pronounced. Saturation thresholds for those schemes were also significantly lower than for DRL, indicating the overall superiority of the DRL topology. Latency and throughput results for synthetic benchmarks are shown in figures 4.1 and 4.2. Dotted vertical lines mark the estimated saturation threshold, calculated using a linear regression model against the ideal throughput, which is shown as a black diagonal dotted line (throughput = in-

jection rate × number of nodes), and confirmed with asymptotic fitting of the latency graphs.

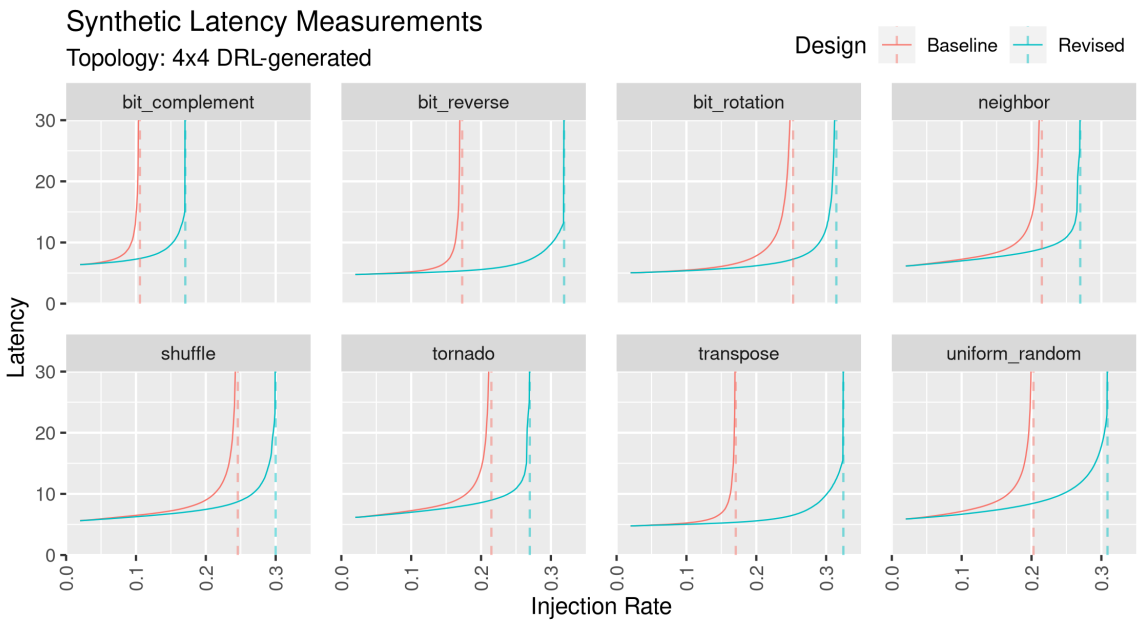

Figure 4.1: **Synthetic Benchmark Latency Results**. Dotted lines show calculated saturation thresholds. Asymptotic behavior with tight curvature near the saturation threshold is indicative that measurements are close to true steady state conditions as $t \to \infty$. In all cases, the revised scheme has a higher saturation threshold, between 22 and 90% higher, with an average of 48% improvement. Latencies prior to saturation are similar for both schemes, as expected.

For all of the benchmarks, the revised scheme outperforms the baseline scheme in global latency and throughput at all injection rates. The revised scheme also generally has a much higher saturation threshold, and a sharper transition to the saturation region. Especially evident in the throughput graphs, the smoothness of baseline's throughput at the saturation threshold indicates early triggering of its

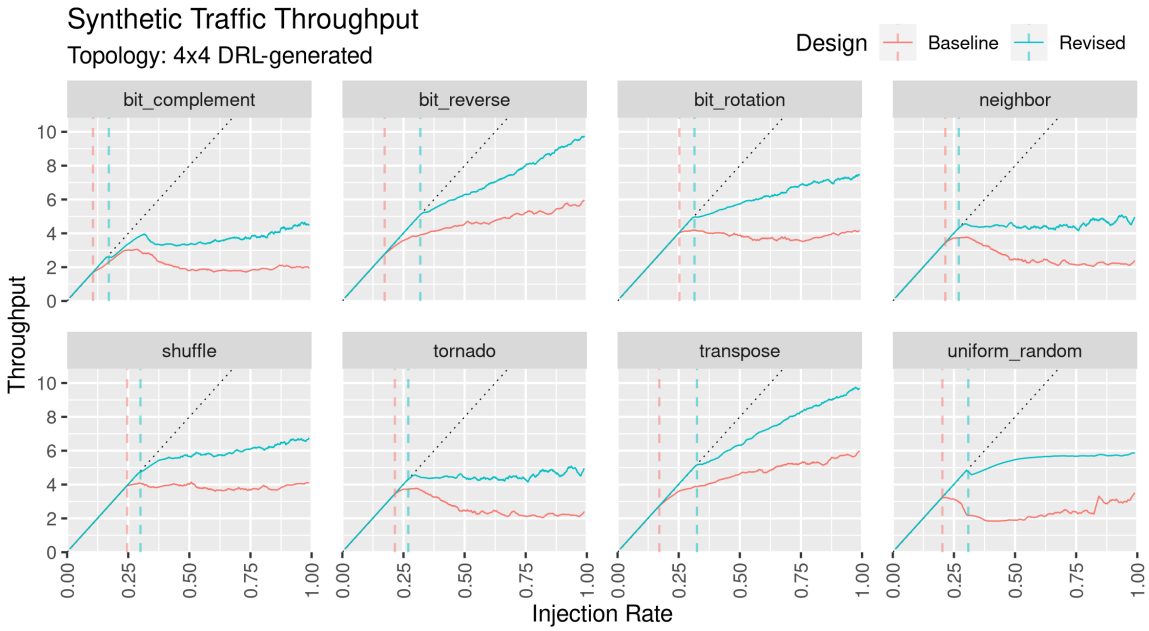

Figure 4.2: **Synthetic Benchmark Throughout Results**. Dotted vertical lines show calculated saturation thresholds. Diagonal dotted line shows perfect behavior without congestion. Sharp breaks at saturation indicate results reliably approximate steady-state conditions as $t \to \infty$. In all cases, the revised scheme outperforms the baseline scheme, both in throughput, and saturation threshold.

recovery schemes and their associated performance degradation from ideal behavior, as well as its susceptibility to congestion, described in more detail in the next section. In contrast, the revised scheme shows a sharp transition into the saturation region in all cases except for bit complement, which appears to partially saturate for some nodes shortly before fully saturating, for both designs.

Throughput climbs in the saturation region for several benchmarks due to these schemes having some number of nodes which produce only self-addressed traffic. Self-addressed packets are immediately ejected at the start of the cycle on which they

are dispatched, and are counted in the simulation statistics as zero-latency packets. Because there are four nodes in both bit reverse (binary addresses 0000, 0110, 1001, 1111) and transpose (binary addresses 0000, 0101, 1010, 1111) that send packets to themselves, throughput climbs proportionately to the number of self-addressing nodes in the saturation region for these benchmarks.

4.1 Over-Compensation in Recovery Schemes

The livelock and starvation schemes of both the baseline and revised designs are recovery-only schemes that are triggered when the interface detects possible livelock and starvation conditions. These schemes essentially all function in some way to exert back-pressure on injecting nodes, which throttles the network. Ghost (or dummy) packets in the starvation schemes prevent intervening nodes from absorbing bubbles, reducing their injection capabilities. Reserving an ejection port for a tagged packet in the livelock scheme reduces the overall rate of ejection, but also distributes the resource fairly between loops so that congestion is more evenly spread across the network. Overly aggressive throttling schemes can seriously hinder performance of a network when they are triggered too often and can cause early saturation by capping maximum throughput. On the other hand, research has shown that carefully tuned throttling can produce paradoxical performance improvements by reducing higher order packet-interaction effects of congestion under high injection rates [9].

In my results, the livelock scheme never triggered for either the baseline or the revised design, but the starvation scheme did trigger frequently at elevated injection

rates. This indicates that packets are always ejected in a timely manner, but the bubbles produced in their absence are immediately consumed for injection of new packets when the network is congested, starving downstream nodes. Relative starvation scheme activation is shown in figure 4.3. The activation rate is measured as number of flits tagged in either design. For the baseline design, only the head flit of a single packet is tagged, while the revised design tags flits continuously until starvation ends. Because of this difference, the revised scheme tags approximately 100 times as many flits at its maximum rate compared to the baseline scheme, and values were normalized to the maximum tag rate measured for each scheme so that the qualitative differences between the schemes are more apparent.

Ideally, neither scheme should activate before true saturation is reached, because this would reduce performance and reduce the saturation threshold, and both schemes should activate consistently post-saturation to recover performance in the saturation region by enforcing congestion control.

As seen in 4.3, the baseline scheme begins to activate much earlier than the actual saturation threshold. The effect of this can be seen in 4.2 with early performance roll-offs for several benchmarks. The baseline scheme also has non-uniform activation in the saturation region, both in terms of random noise, and also a clear trend in some of the graphs, such as under the uniform random traffic pattern, where it drops off sharply towards the right side of the graph. The high cost of the scheme can be seen by referring back to 4.2 where the throughput drops considerably coinciding with a spike in starvation scheme activation for that particular injection rate. It's likely that this scheme has a considerable effect on network performance and degrades it to the

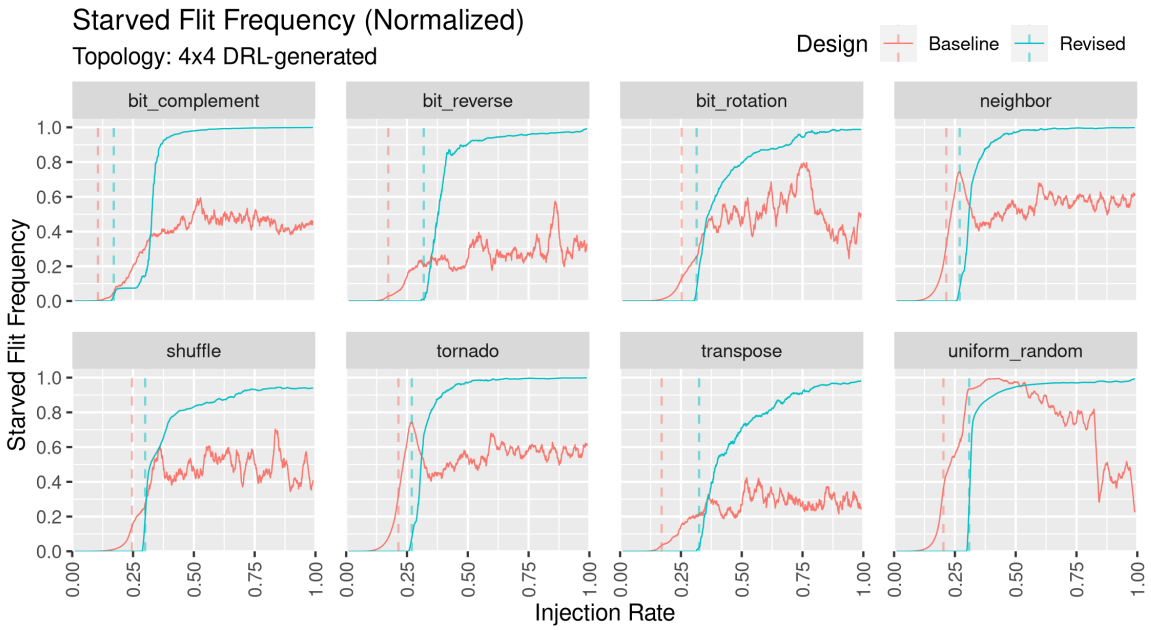

Figure 4.3: **Synthetic Benchmark Starvation Results**. Shows relative frequency of starvation scheme activation, normalized to the maximum activation rate for each design.

point that self-reinforcement causes the scheme to lock itself into an activated state.

In contrast, for all of the different benchmarks, the revised starvation scheme does not trigger until the saturation threshold is reached and consistently triggers after starvation for all but the bit-complement benchmarks. The two-step shape of the bit-complement benchmark result is in-line with the throughput result that shows early saturation of some of the nodes on the network prior to total network saturation later on; that is, only some nodes are regularly starved until a second saturation threshold is reached past the initial one and all nodes start to starve.

4.2 Hysteresis

When the network returns to sub-saturation injection rates from the saturation region, its injection buffers remain backlogged at first, and the network continues to behave as if it was still servicing a saturating injection rate. Thus, it will maintain its diminished throughput and continue triggering the recovery schemes until the backlogged injection buffers are emptied. Keeping in mind that the injection buffers will only empty when the injection rate drops below the current throughput, it can be inferred that the network should exhibit hysteresis on returning from saturation. It will not return to normal operation until the injection rate drops below the saturation throughput, a point hereafter referred to as the "lower saturation threshold", which may be some amount less than the "upper saturation threshold". This is shown in figure 4.4 by the arrow labeled "Hysteresis Effect". Because of this effect, it is important that recovery schemes do not trigger outside of the saturation region, and also that they have a minimal negative impact on performance, so that the lower saturation threshold stays close to the upper saturation threshold. Otherwise, network performance will remain degraded in the region between upper and lower saturation thresholds following periods of burst traffic above the upper saturation threshold.

In the throughput results for the synthetic benchmarks, shown in figure 4.2, the expected performance dip is evident in several of the subplots. For the revised scheme, this performance loss is barely evident, only discernible under uniform random and bit complement benchmarks. The baseline scheme has early roll-off indicative of early starvation scheme activation, and pronounced dips in the saturation region coinciding

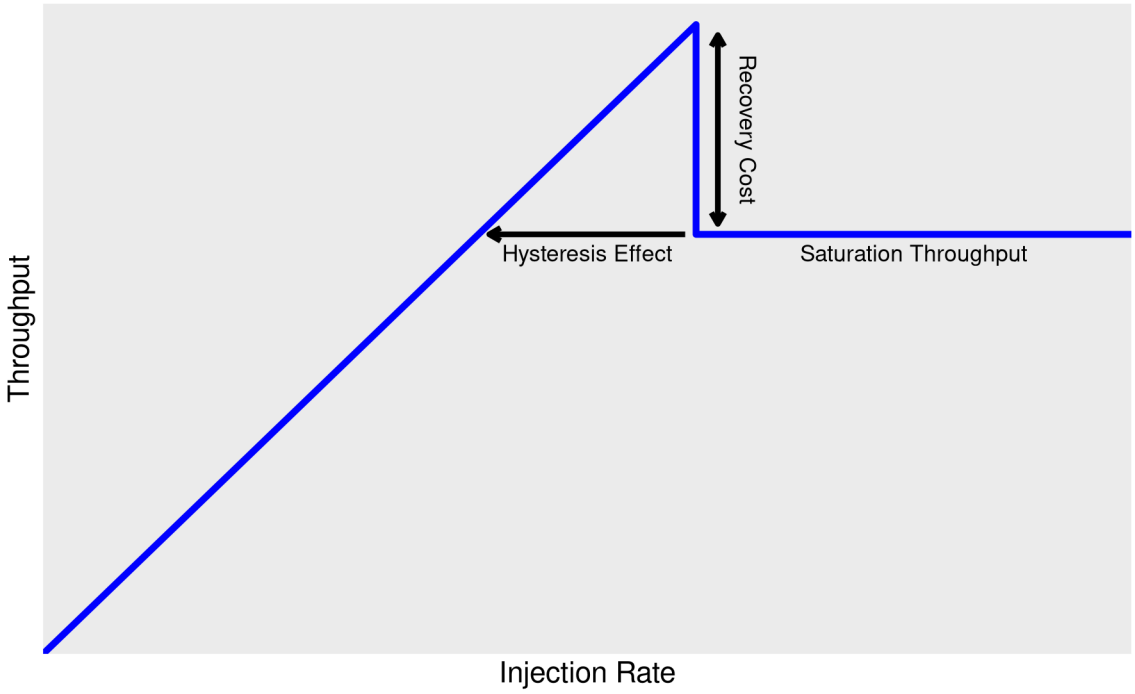

Figure 4.4: **Hysteresis Effect**. Injection rate must drop to equal saturation throughput to return to normal operation.

with the starvation scheme trigger rate, as discussed previously. Overall, the revised scheme's throughput in the saturation region is only marginally less than the throughput at the upper saturation threshold, indicating the hysteresis effect will be weak to non-existent. In contrast, the throughput of the baseline scheme drops considerably after the upper saturation threshold is reached in several benchmarks, indicating very wide lower-to-upper saturation threshold gaps. For uniform random, this gap is greater than 50%, meaning the lower-saturation threshold is less than half the upper saturation threshold for the baseline scheme. Effectively, once the starvation scheme in the baseline scheme triggers, it reduces performance so much that it has a self-

reinforcing effect even as injection rate drops below the saturation threshold. Thus, I expect the baseline scheme to be much more susceptible to performance degradation from periodic burst traffic patterns compared to the revised scheme, which may be more robust.

4.3 Deadlock Scheme Effects

Because the deadlock scheme is a structural change to the network, I expect to see the same behavior at low injection rates where network load is low, and resources are never fully allocated; instead I consider the saturation threshold and behavior in the saturation region, when resources are under heavy use. I mainly attribute the large shifts in the saturation threshold seen in the synthetic benchmarks to the deadlock scheme's reallocation of resources, because the starvation scheme doesn't consistently trigger until after saturation. These performance increases are expected, because the revised deadlock scheme has twice as many EXBs in its pool compared to the baseline design, so it has much more ability to keep injecting packets when under load, and to fully utilize its loops. Furthermore, the better throughput under saturation is largely related to the delayed saturation thresholds of the revised scheme, and due to the higher resource availability in the extension buffer pool.

4.4 Full System Results

Figure 4.5 shows the reported execution time for the region of interest of each of the PARSEC benchmarks running under full system. The revised scheme outperforms the baseline scheme for all benchmarks shown, except canneal (66% as fast) and freqmine (98% as fast). The best result was a speed increase of 36% for streamcluster. Overall, these results show that the revised scheme's performance is as good or better in most cases compared to the baseline under full-system conditions. This is as expected, because full-system benchmarks generally stay within low injection rate regions that show idealized network behavior under synthetic benchmarks. Discrepancies likely reflect on the increased extension buffers available at each node, better equipping individual nodes to handle burst traffic.

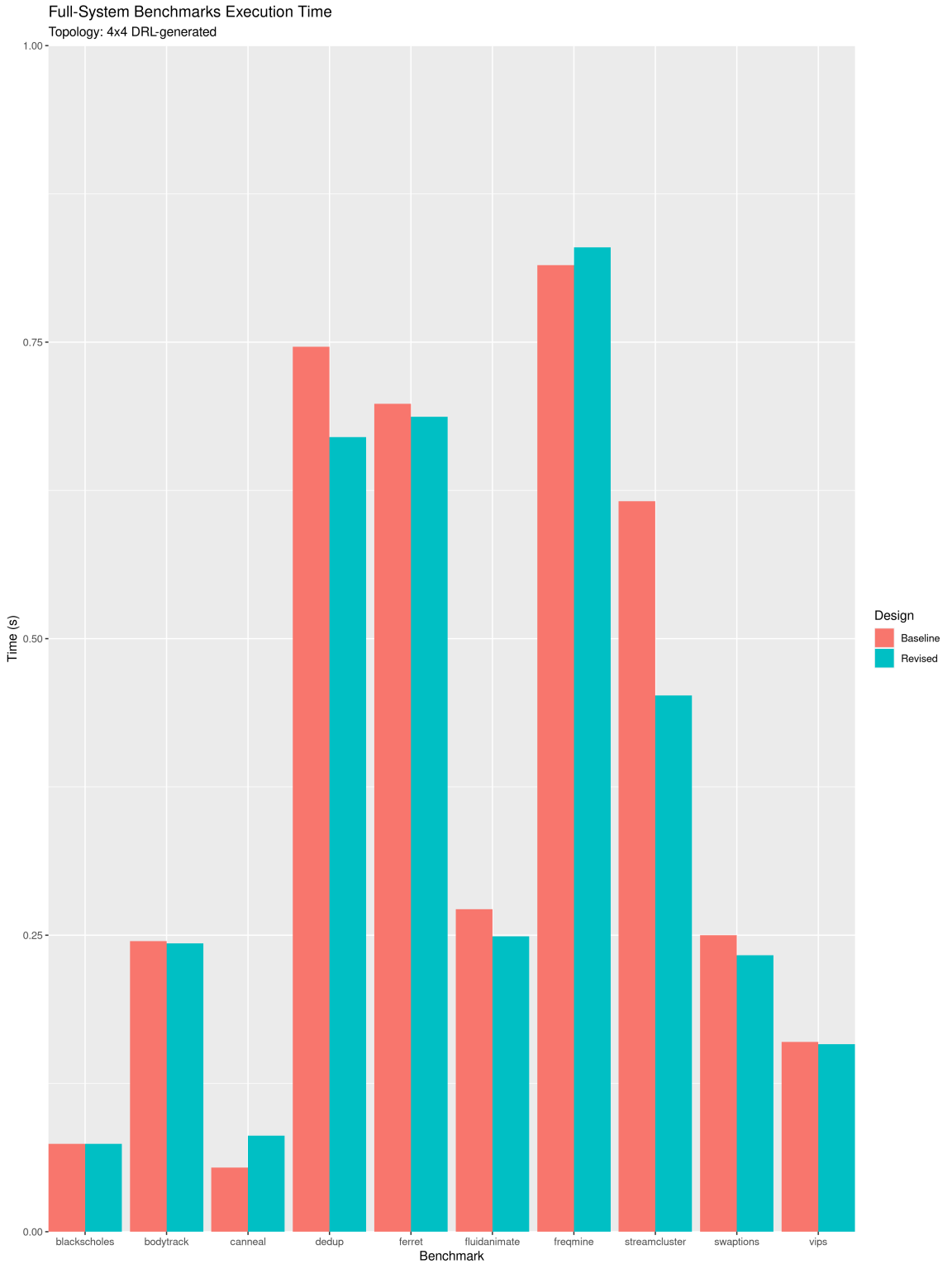

Figure 4.5: **Parsec Benchmarks Execution Time**. Smaller values are better. The revised scheme outperformed the baseline scheme for all benchmarks except canneal and streamcluster.

Chapter 5: Conclusion

Changes in manufacturing processes and design principles for chip multiprocessors are driving a need for the development of next generation networks on chips to replace aging router and ring paradigms. Routerless was recently introduced to serve the needs of new chip designs with a lightweight, scalable interface that is able to efficiently harness newly available wiring resources while freeing much of the silicon normally devoted to network routers.

While other research has focused on optimizing the network topology for routerless networks using deep reinforcement learning, no research has yet been published on handling network abnormalities. In this work, I developed a theoretical basis for analyzing network abnormalities which I then used to demonstrate the inadequacy of existing proposed schemes for overcoming livelock, deadlock, and starvation in routerless networks. I then presented three new schemes and validated them for correctness under the same theoretical treatment.

Both synthetic and full-system simulation results demonstrate that the newly revised schemes I have presented dramatically outperform the baseline schemes, in addition to their theoretical correctness. The performance of the revised schemes is qualitatively uniform across traffic patterns and in-line with *a priori* expectations of idealized network behavior. In contrast, the baseline design shows irregular, noisy performance characteristics. This indicates the effectiveness of the revised design

both at preventing livelock, deadlock, and starvation and in improving performance both qualitatively and quantitatively over the baseline scheme. Results also indicate that the revised scheme is less sensitive to transient congestion due to reduced cost of starvation scheme activation, resulting in a smaller hysteresis effect on return from saturation injection rates.

Routerless networks introduce a vast, unexplored design space to networks on chip; many areas of research on this topic remain, including further investigation and validation of the results and effects discussed in this work. It is my hope that my implementation of the routerless network interface in gem5 will accelerate research efforts in this direction.

Bibliography

[1] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj Kumar Jha. Garnet: A detailed on-chip network model inside a full-system simulator. *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 33–42, 2009.

[2] F. Alazemi, A. AziziMazreah, B. Bose, and L. Chen. Routerless network-on-chip. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 492–503, Feb 2018.

[3] L. A. Barroso and M. Dubois. The performance of cache-coherent ring-based multiprocessors. In *International Symposium on Computer Architecture*, 1993.

[4] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[5] Christian Bienia and Kai Li. *Benchmarking modern multiprocessors*. Princeton University New York, 2011.

[6] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, Bin Liu, A. Tran, E. Adeagbo, and B. Baas. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *Symposium on VLSI Circuits (VLSI-Circuits)*, 2016.

[7] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, Bin Liu, A. Tran, E. Adeagbo, and B. Baas. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *IEE Symposium on VLSI Circuits (VLSI-Circuits)*, 2016.

[8] Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pages 746–749. ACM, 2007.

[9] Kevin Kai-Wei Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu. Hat: Heterogeneous adaptive throttling for on-chip networks. In *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, pages 9–18, 2012.

[10] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 684–689, New York, NY, USA, 2001. ACM.

[11] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. A 5-ghz mesh interconnect for a teraflops processor. *Micro*, 2007.

[12] Jason Howard, Saurabh Dighe, Sriram R Vangal, Gregory Ruhl, Nitin Borkar, Shailendra Jain, Vasantha Erraguntla, Michael Konow, Michael Riepen, Matthias Gries, et al. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 2011.

[13] Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh. *On-Chip Networks: Second Edition*. Morgan & Claypool Publishers, 2nd edition, 2017.

[14] T. Lin, D. Penney, M. Pedram, and L. Chen. A deep reinforcement learning framework for architectural exploration: A routerless noc case study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 99–110, Feb 2020.

[15] S. Liu, T. Chen, L. Li, X. Feng, Z. Xu, H. Chen, F. Chong, and Y. Chen. Imr: High-performance low-cost multi-ring nocs. *IEEE Transactions on Parallel and Distributed Systems*, 27(6), 2016.

[16] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur,

Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+, 2020.

[17] John D. Owens, William J. Dally, Ron Ho, D. N. (Jay) Jayasimha, Stephen W. Keckler, and Li-Shiuan Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, September 2007.

[18] Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Priya Iyer, Arvind Singh, Tiju Jacob, et al. An 80-tile 1.28 tflops network-on-chip in 65nm cmos. In *IEEE International Solid-State Circuits Conference*, 2007.