CONGESTION AWARE ADAPTIVE ROUTING FOR NETWORK-ON-CHIP COMMUNICATION

by

Stephen Chui
Bachelor of Engineering
Ryerson University, 2012

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada

Author Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Congestion Aware Adaptive Routing For Network-On-Chip Communication

Stephen Chui
Master of Applied Science, 2016
Electrical and Computer Engineering
Ryerson University

Abstract

Network-On-Chip (NoC) has surpassed the traditional bus based on-chip communication in offering better performance for data transfers among many processing, peripheral and other cores of high performance embedded systems. Adaptive routing provides an effective way of efficient on-chip communication among NoC cores. The message routing efficiency can further improve the performance of NoC based embedded systems on a chip. Congestion awareness has been applied to adaptive routing for achieving better data throughput and latency.

This thesis presents a novel approach of analyzing congestion to improve NoC throughput by improving packet allocation in NoC routers. The routers would have the knowledge of the traffic conditions around themselves by utilizing the congestion information. We employ header flits to store the congestion information that does not require any additional communication links between the routers. By prioritizing data packets that are likely to suffer the worst congestion would improve overall NoC data transfer latency.

Acknowledgement

I would like to express my sincere appreciation to my supervisor Dr. Gul N. Khan for all his guidance and support while pursuing my graduate studies in the MASc program. I am grateful for his contribution of many hours of his time and efforts to me during the duration of my studies. I would also like to acknowledge NSERC and the Department of Electrical and Computer Engineering at Ryerson University for their financial support. Finally, I would also like to extend my thanks to my parents, Mr. and Mrs. Chui for their supportive environment as I pursue my graduate studies.

Table of Contents

List of Tables

List of Figures

Lists of Abbreviations

BE	Best Efforts
BW	Buffer Write
CAAR	Congestion Aware Adaptive Router
DAR	Destination Adaptive Routing
DBAR	Destination Based Adaptive Routing
DMesh	Diagonal Mesh
DOR	Dimension Order Routing
DyAD	Dynamic-Adaptive-Deterministic
DyXY	Dynamic XY
FIFO	First In First Out
GCA	Global Congestion Awareness
GT	Guarantee Throughput
HoL	Head-of-Line
IC	Integrated Circuit
LT	Link Transversal

MUX	Multiplexor
NoC	Network-on-Chip
OE	Odd-Even
QoS	Quality of Service
RAM	Random Access Memory
RC	Route Computation
RCA	Regional Congestion Awareness
RR	Round Robin
SA	Switch Allocator
SoC	System-on-a-Chip
SRAM	Static Random Access Memory
VA	VC Allocator
VC	Virtual Channel
VCID	Virtual Channel ID
VLSI	Very Large Scale Integration
Xbar	Crossbar
XT	Crossbar Transversal

Chapter 1

Introduction

1.1 NoC Systems

Embedded systems on chip in the recent decade have grown substantially utilizing many cores on a single chip known as embedded System-on-a-chip (SoC). It is important that data within the SoC, all cores have access to the desire resources while maintaining a data load balanced transmission links. Traditional bus-based interconnection has been able to allow high speed data transfers within a small SoC. However as the SoC scales up in size in today's most demanding applications, NoC is proven to provide better balance between traffic loads and data access for every core. [1] NoCs are expandable to allow communication for very large SoCs. It is not uncommon to see more than 256 cores in a NoC system, which is impossible to handle on a bus-based system.

NoC based systems is a new strategy for data communication within the SoC. NoC performance depends on topology, data link width, traffic patterns, routing mechanisms and router arbitration. These parameters can be prioritized to improve performance, area of the design layout or power dissipation of the SoC. Depending on the chosen design requirement, the connections between routers and cores of a NoC can differ significantly.

NoC performance is important in high data throughput applications. It is ideal to reduce congestion and latency for all the data transfers within the NoC. Unfortunately, by increasing the number of cores within the NoC, congestion increases proportionally to the NoC size. Therefore, it is necessary to manage traffic effectively for reducing congestion within the NoC to achieve the best performance as the NoC grow in size.

1.2 Motivation

NoC development is rapidly advancing thanks to improved manufacturing technologies within the VLSI field allowing more processing and other cores to be embedded in a SoC. As multi-core systems become the norm of this decade's technology, there is a need to allow more cores to communicate within the SoC. The growing number of cores will contribute to additional traffic to the expanding size NoCs which will increase congestion. There is a need to reduce congestion and improve performance at the same time for larger size NoCs. This leads to the needs of conducting research and developing better NoC routers that are adaptive and handle the increasing demands of larger NoC.

As NoC usage grows within the industry, it has become necessary to have off the shelf NoC models. A mesh topology NoC design is very common for large number of application and it has high scalability for expansion. It is important to have a NoC design that supports high performance in terms of throughput. By introducing congestion awareness based adaptive routing **[**1**,**2**]**, performance gain has occurred with the addition of some hardware components. More importantly, the additions of these specialized components are done within the NoC thus any developer utilizing this platform would not have to design a different interface to take

advantage of congestion awareness. This will also help new developers to employ adaptive NoCs an alternative to traditional techniques such as a bus based system.

Surely the last decade, there has been ongoing research for a better routing system for mesh topology NoCs. It started off with turn based models such as Odd Even (OE) routing [3] which can be applied to adaptive routing. Further along, researchers have investigating routing flexibility to reduce loading on ports in one axis. Deterministic routing such as Dimension Order Routing (DOR) (e.g. XY Routing) has been utilized for simplicity and the ability to avoid deadlock and livelock situations [1]. The idea of a better routing algorithm is needed to choose between the x-axis and y-axis which was pursued to improve loads within the NoC. Adaptive routing utilized congestion data of neighbours first. As this research area matured, adaptive routing utilized more data from routers beyond its neighbours. A steady flow of data was needed to achieve regional congestion awareness for adaptive routing [4]. Destination based routing has shown to improve latency as congestion is more accurately known to the destination **[**5**,**6**]**.

We investigate, how much congestion awareness information is necessary for a NoC with congestion awareness to operate reliably. How congestion related data should be transmitted to relevant routers. Moreover, how congestion data can be utilized by other components of the router to improve latency?

1.3 Contribution

The main objective of the research presented in this thesis work is to improve performance of the NoC routing by congestion awareness information received. Previously congestion awareness is mainly used in routing decisions. This work extends congestion information usage to other parts of the NoC routers such as to improve allocation and buffering capabilities.

Secondly, most newly developed NoC microarchitectures are often tested with average latency only. Although average latency provides a good measure of performance, it does not outline the worst case scenario which is critical for embedded SoC systems. Maximum latency performance is analyzed to understand their effects in a congestion aware adaptive router (CAAR).

In the thesis, attempts to improve throughput and reduce latency are made by prioritizing packets under congested situation. Packet prioritization during congestion is proposed to improve latency between packets that are the furthest away between the source and destination cores. This would improve the overall latency within the NoC and allow these packets to suffer lower latencies. Moreover, VC resources would be freed up for other packets to utilize the buffer space. In addition to prioritizing packets, this method can be extended to the application of Quality of Service (QoS). This allows highly sensitive packets to be prioritized under congestion ensuring on time delivery without delaying other packets under non congested operations.

1.4 Thesis Structure

This thesis covers the concept of NoC communication, our congestion awareness methodology, experimental setup and simulation results. Chapter 2 provides an overview of the NoC and how topology, traffic patterns, router architecture and design will affect performance. Moreover, adaptive routing and congestion awareness techniques are discussed in details such as the different types of adaptive routing and how congestion data are transmitted within the NoC. Chapter 3 explains the methodology for prioritizing packets and how the allocators determine and select prioritized packets under congestion. Chapter 4 provides the developed simulator and setup for the experiment of improving NoC performance and reducing congestion within the NoC. It describes the development of the simulator in SystemC. Chapter 5 illustrates the results of the experiment conducted for improving congestion the NoC. Finally, Chapter 6 concludes this thesis with recommendations and future work.

Chapter 2

Congestion Aware NoC

In this chapter, we will discuss important components of the NoC and its router for congestion awareness. This chapter provides a description of data flow, router architecture, flow control, traffic generation, adaptive routing and congestion awareness. We will offer an insight of the advantages and disadvantages of different methods available for NoC design. Efforts are made to highlight various technique employed to improve performance of the mesh topology NoC. Moreover, we will describe the techniques relating to congestion awareness hardware, modifying the mesh topology and the changes made to the router's microarchitecture.

2.1 NoC Topology

Topology is the interconnection structure of the NoC that consists of routers, links and cores. There are many possibly topologies for NoC including mesh, torus, ring, star, cube and etc. **[**7**,**8]. NoC topology can expand into many dimensions. Each topology has its own advantages and disadvantages such as flexibility, reliability, number of links, routing latency and complexity. Performance would differ depending on topology since they are connected differently with various numbers of links [**7**]. A simple topology such as ring topology has very

simple routing mechanics but the number of routers is limited since latency will increase substantially if too many routers are added to the NoC. Mesh topology is generally chosen as it has many benefits such as multiple paths to every router, supports adaptive routing and fits many applications.

Figure 2.1 shows how a star topology is organized. This topology designates the centre router R0 as the main router. If router R0 is congested, it affects all router to router transfers but allows local cores within the slave routers R1-R5 to continue functioning. Each slave routers can be further expanded into branches but in the case where cores need to communicate with other branches, significant congestion would be created. The worst case scenario is when R0 fails as there is no backup link causing the system to total fail. Figure 2.2 illustrates a ring topology NoC. Unlike a start topology, the ring topology has two directions to route its data. This topology handles fault tolerance better than a star topology. The ring topology still suffers with the congestion issues. In the case where a router is congested, it creates a backlog of data holding up all data transfers behind it. The mesh topology resolves these issues with many alternative links between routers allowing multiple paths from a source to destination.

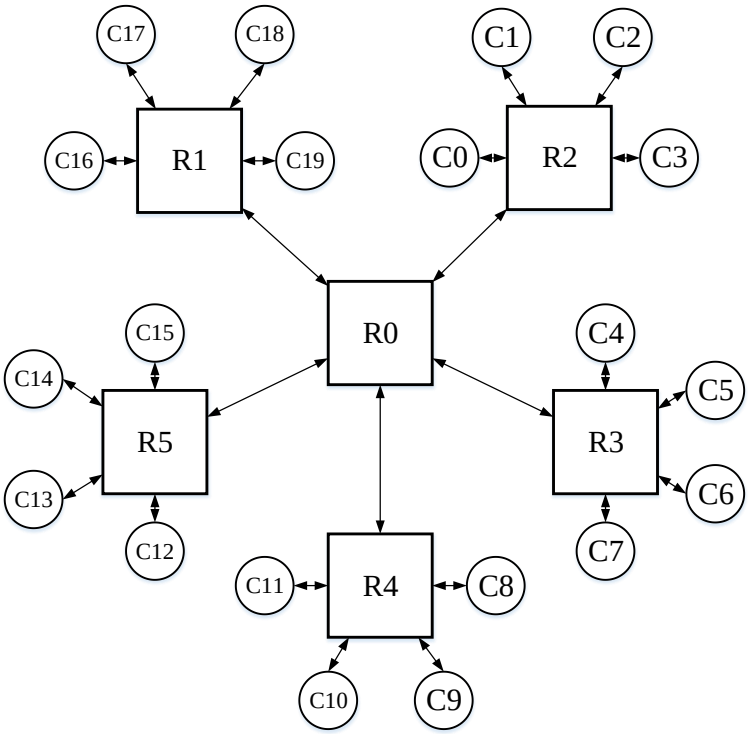


Figure 2-1 - Star Topology NoC

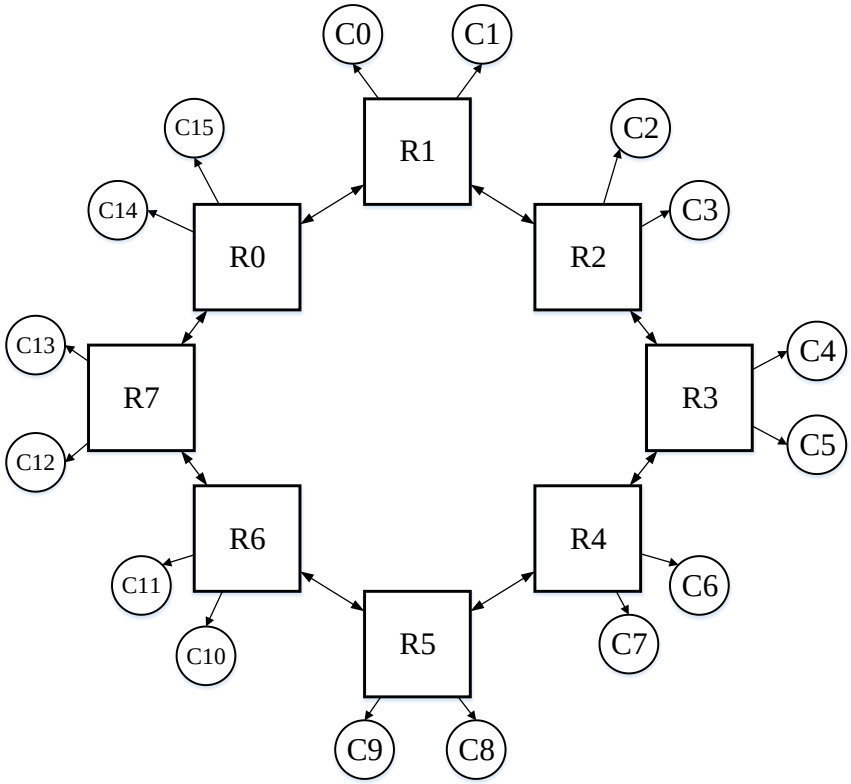


Figure 2-2 - Ring Topology NoC

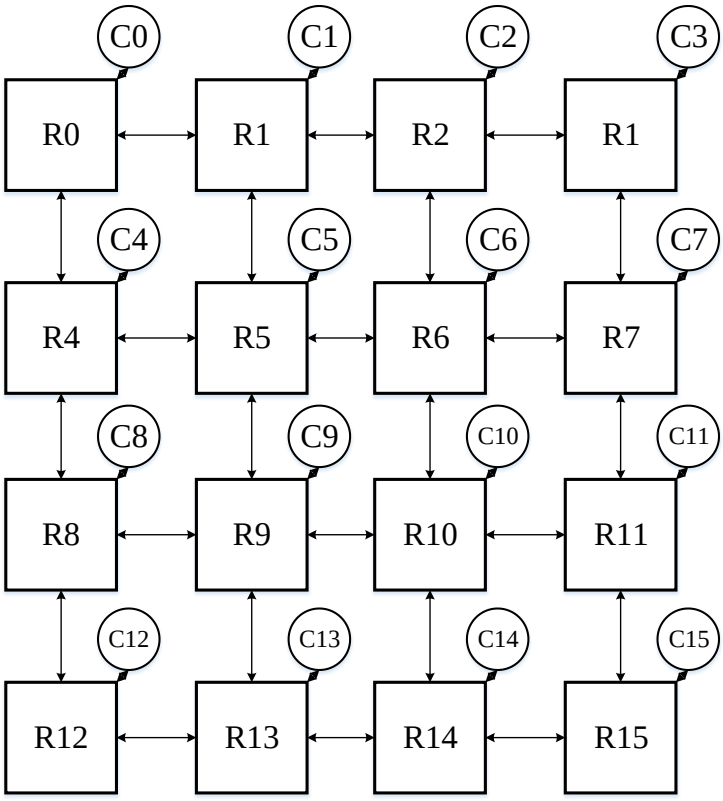


Figure 2-3 - Mesh Topology NoC

Figure 2.3 shows a typical 4x4 mesh topology NoC. Please note the difference between mesh and the previously illustrated topologies where each router only has a single connection to its core. This improves performance by reducing demand on each router and allowing multiple paths to a destination at the expense of more routers. Torus topology is an extension of this topology that also connects the routers on the edges among each other. The mesh topology can be stacked together to form higher dimension NoCs such as 3D cube topology.

2.2.1 Modified Mesh Topology

The mesh topology is a standard topology where routers connected in all four cardinal directions as shown in Figure 2.3. It is easily scalable and thus it has been used in many research works as well as physical applications [9]. Mesh topology is simple and adaptive to many NoC designs making it an attractive choice. Although the mesh topology offers a lot of flexible as a NoC topology, it has a limitation to transmit data diagonally or directly beyond the neighbouring routers. Qian et al. suggests that the mesh topology can be broken into regions of 4x4 and a hub router can be added to the NoC which can dynamically reconfigurate itself to connect to surrounding routers allowing packets to be transmitted diagonally to reduce latency [10]. Another implementation by Wang et al. modifies the mesh topology into a diagonal mesh (DMesh) which has additional diagonal links connecting each router on the NoC [11]. Both implementations show improvements over the traditional mesh topology at the cost of additional links and increase complexity of each router. Crossbars have additional contention from more input and output ports which can potentially reduce performance.

2.3 Data Transmission

NoC data transmission can be setup either by employing circuit switching or packet switching **[**8**,**12**]**. Each method has its own advantages and disadvantages. Circuit switching requires a direct path to be setup from the source to destination cores to allow the data flow from the source to destination. Unassigned cores suffer longer waiting for the grant of physical channel for data transmission. Packet Switching is a method where the message data is divided into packets that are further divided into flits. With a smaller data unit such as flits, data can be buffered easily between multiple routers of the NoC. This allows the NoC to function as

wormhole routing instead of store-and-forward. Store-and-forward requires all the flits of a packet to be transmitted and stored in a router first before it can be transmitted to the downstream router, which requires a larger amount of buffer space.

The data organization for data transmission in a packet switching NoC is commonly known as a message. In a traditional bus based NoC system, the source and destination cores are first determined before transistor switching for data transmission. On the other hand, a packet switching NoC requires a source and destination to be embedded within the message since multiple messages can travel in the NoC at any given time. Since the size of a message is considerably large, it can be broken down into packets to allow more flexible routing in the NoC.

2.3.1 Packet

A packet is the basic data organization unit within the NoC. A packet consists of one or multiple flits containing information for routing purposes and the actual data that is necessary to be transmitted from a source to the destination core. A packet typically consists of a header flit, some body flits and a tail flit [9]. Some packets are just one flit long while others can be longer than 12 flits. Packet size has an effect on routing and virtual channel (VC) occupation at each buffer. In the case of wormhole routing, a packet will lock up an output port of the NoC router until all the flits are transmitted to the downstream router. Similarly in a VC based routing NoC, each packet arriving at a router will be assigned a virtual channel ID (VCID) and it is locked to that specific VC until all the packet flits are transmitted.

2.3.2 Flits

Flits are a sub-unit of a packet. All the flits have an indication of head, body or tail flit and a VCID in the case of VC routing. A header flit contains has the routing information such as destination ID and possibly source ID to identify where the packet has come from. In the case of look-ahead routing, both output direction options are embedded in the header flit. Furthermore, congestion information can also be transmitted in a header flit. Body and tail filts usually just contain an indication bit for body or tail flit, the VCID (if necessary) and general data. Tail flits are similar to body flits but has a special bit to signal to the router so that it can release the output port (wormhole) or VC (VC routing). Figures 2.4 and 2.5 illustrate an example of the header and body flit organization for a router that supports adaptive routing. In this example, each flit is 128 bits long. The destination ID is required for routing while the source ID may be necessary for the application. Options 0 and 1 are the two possible directions for adaptive routing. The congestion bit and data are used to transfer congestion data in the header flit.

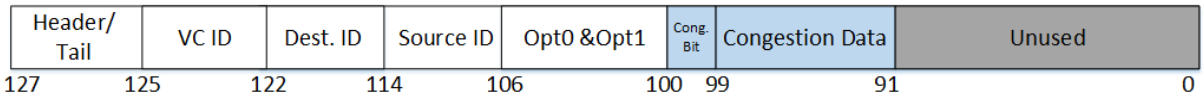


Figure 2-4 - Header Flit Organization

Figure 2-5 - Body/Tail Flit Organization

2.4 Router Architecture

The NoC router consists of three main components consisting of a buffer, arbiter and crossbar switch. The arbiter allocates packets stored in the buffer using the crossbar switch. The design of each component depends on the desired NoC performance, chip area and power consumption constraints. A more complex router design improves performances at the expenses of additional area and power consumption.

2.4.1 Buffer

The NoC Buffer is a key storage component in the NoC router. The buffering system in each NoC router allows the NoC to maintain a free-flowing data link between routers to reduce Head-of-Line (HoL) blocking [14]. Buffers are usually implemented as First-In-First-Out (FIFO) based data structure utilizing Static Random Access Memory (SRAM). The size of each buffer should be approximately the average size of each packet. This would allow most packets to be able to store completely in one buffer without having half the packet queued in the upstream router.

2.4.2 Virtual Channels

Buffers often are filled while the packet awaits allocation during the transmission of another packet in the downstream router. This often causes additional congestion due to HoL Blocking. To vastly improve congestion in the NoC, virtual channels are introduced to allow a

flit of multiple packets to be stored at the input port of each router [15,16]. This allows another packet to transmit in the case that the previous packet awaits allocation. Additional VCs would utilize additional buffer storage and chip area and increasing the power consumption. The number of VCs in each buffer should be chosen to maximize performance while keeping the integrated circuit (IC) area usage as low as possible. Adaptive routing requires more VCs as compared to traditional deterministic routing to ensure performance is maintained [17,18,19]. An additional VC for packets to escape in a deadlock situation is needed. This would be further explained in the Adaptive Routing section.

Normally all the VCs have the same priority which allows the Round Robin arbitration scheme to select the next VC. Although it allows fairness, but there is no guarantee of minimum latency. There are architectures that support prioritized VCs. An example is Æthereal, which provides two different types of VCs, best effort (BE) and guarantee throughout (GT) VCs [20]. GT VCs are always prioritized to ensure a lower latency. Although these architectures allow latency to be minimized, it requires distinction of two different types of VCs that maybe wasted when they are unoccupied.

2.4.3 Arbiter

The arbiter acts as the main controller of the router consisting of the routing mechanism, VC allocator and switch allocator. These three components are important to determine the next hop for all the arriving packets along with the VCID for the downstream router and assigning which flit will gain control to transverse the physical link to the downstream routers. The arbiter also contains the VC allocator and switch allocator. One of the most basic allocator is the round

robin allocator. It provides fairness and it is simple to be implemented. Simplicity comes at the cost of less efficiency. Round robin does not always provide the best throughput [21,22]. A modified round robin scheme known as iSLIP also requires the input VC requests to be selected to maximize throughput [23]. Round Robin may not provide the best selection as it is based on sequential order. Wavefront allocator which grants inputs to outputs based on a matrix allows better matching [2]. Furthermore, Becker et al. introduces spare VC allocation which separates all the packets into two classes. A packet would be assigned randomly to one of the classes at the injection instances. Packets cannot change classes while being transmitted through the NoC. With less VCs to grant, this allows lower contention between different VCs which improves the chip area, lowers combination logic delays and saves on power and energy consumption [22].

2.4.4 VC Allocator

In VC based NoC Routing, the VC allocator assigns virtual channel IDs (VCID) for every packet being transmitted at every router. This allows the packet to be buffered in one of many VC buffers at the downstream router. If a packet is unable to be assigned a VCID, the packet must wait for the next cycle and the VC allocator attempt again. This process repeats until the packet receives a VCID. The VC is released once the tail flit is transmitted.

2.4.5 Switch Allocator

The switch allocator is another component that allocates incoming flits to the output ports. Switch allocation occurs at every cycle in which it will determine the flit from the input

port will be selected for the output port in the case of contention [22]. Round Robin (RR) mechanism is usually employed to determine the winner. RR offers fairness and avoids starvation for all the fits. Some of the switch allocator designs use a modified RR scheme knows as iSLIP to improve throughput within each router [23].

2.4.6 Speculative Switch Allocator

Normally, VC allocation must successfully assign a VCID for the incoming packet before the switch allocator allows any flit of that packet to be assigned for crossbar switch transversal. Unfortunately it takes one complete clock cycle for a packet to be assigned a VCID which adds latency to the packet at each router level. An additional switch allocator is added to the arbiter to allow new packet arrivals without an assigned VCID to allocate an output port. This allows the VC allocation and switch allocation process to occur in parallel to reduce latency [2]. In an event that the switch allocator has a request for the output ports, it will have priority over the speculative switch allocator. An additional switch allocator also requires more IC area and power consumption.

2.4.7 Crossbar

The crossbar switch is a set of multiplexors for each output port direction. In a 2D mesh topology NoC, there are usually five sets of 5-to-1 multiplexers in the crossbar. The crossbar is controlled by the switch allocator. A modified mesh topology such as DMesh can have up to nine inputs and outputs. To avoid starvation due to large number of inputs, some of the inputs are

removed by using two sets of crossbars [11]. Generally, the crossbar switch requires significant IC area as each multiplexor requires bus inputs with the width of a flit.

2.5 Data Flow Control

Flow control ensures that complete packets are delivered reliability from source to destination cores without missing any flits. As multiple packets can compute for an output port at any routers, buffering space can fill up which will lead to data overflow at the local buffers if data is allowed to flow freely from the upstream routers. To ensure that overflow does not occur, a flow control system has to be carefully designed to prevent overflow but also allow the effective usage of the buffer by using the entire buffer capacity. Flow control can be implemented in many different ways and each implementation has its own advantages and drawbacks. There are mainly two types of flow control as given below.

2.5.1 On-Off Flow Control

On-Off flow control is a simple flow control system that allows upstream router to continuously send flits to the downstream router until the buffer is about to be full. Since there is usually a latency of at least one clock cycle to signal the upstream router to stop sending the flits, the signaling time must occur when the buffer is almost full to prevent overflow. On-off flow control can be implemented easily with simple logic and without the need of any registers. There are major drawbacks for On-Off flow control, especially for congestion awareness NoCs. For any NoCs, On-Off flow control does not always fill up the buffers, especially for shallow depth

buffers where only 60-80% of the buffer is filled on average [14]. Furthermore, in adaptive routing, On-off flow control signals cannot be used to track the congestion accurately. An off signal cannot correctly indicate that congestion is occurring.

2.5.2 Credit Based Flow Control

Credit based flow control is a more sophisticated flow control system that allows the upstream router to track the exact amount of free buffer slots in the downstream router [20]. This ensures that the buffer is fully occupied without any overflow. It ensures that the best usage of each buffer, especially for the shallow buffers type. Credit based flow control requires a register to track the remaining free flit space. The addition of virtual channels means that there are multiple VC buffers per input direction each requiring their own register to track the available free flit spaces. Credit based flow control also have the advantage of using free flit value as an indication of congestion at the downstream router [24].

2.6 NoC Routing

2.6.1 Route Computation

The process to determine the next direction for each arriving packet's header flit is the route computation. One clock cycle is required to determine the next hop direction for each header flit of a packet. In deterministic routing (e.g. XY routing), the next hop directions for all packets are fixed. Deterministic route computation can be simply implemented with a single look

up table. In a more complex routing system such as adaptive routing, dedicated hardware are introduced to allow routing to multiple directions based on traffic conditions to ease congestion (M. Ramakrishna, 2013).

2.6.2 Look-Ahead Routing

Routing computation adds a full cycle to the NoC router's pipeline. As additional latency deteriorates the NoC performance, Look-Ahead Routing is used to hide any additional latency for computing the next hop direction in parallel with another process(Towles, 2003; Bjerregaard, 2006). Unfortunately, it is impossible to compute the next hop direction in parallel to the buffering stage and therefore the routing decision is completed in the previous or upstream router. The upstream router would be aware of the next hop direction based on information communicated from routing unit and it would be able to compute the direction(s) for the packet in the downstream router. This would require a few bits in the packet's header flit to store the next hop direction(s) for look-ahead routing. In the case of adaptive routing, there could be more than one choice for the downstream router. For a mesh topology NoC, there are at the most two possible directions for the next hop direction if the destination core is diagonally located from the current router. Both directions would be stored in the header flit ahead of time to allow the downstream router to select one of the two directions based on the current traffic condition without requiring a full cycle of computation. The direction is already selected one cycle in advance allowing the pipeline to avoid the route computation latency.

2.6.3 Deterministic Routing

Deterministic routing algorithms are used in many NoCs due to its simplicity in terms of hardware implementation, IC area and power constraints. They do not require any additional overhead for decoding the external information offering lower delays within the router and are able to perform all the necessary routing in a NoC.

Dimension Order Routing (DOR) algorithm such as XY or YX routing are commonly employ deterministic routing technique used in mesh topology on-chip networks due to its simple implementation that offers livelock and deadlock avoidance for wormhole routing. DOR algorithm routes a packet in one dimension till it reaches the desired row or column and then turn to continue to the destination. With only one turn, DOR avoids any deadlock or livelock with or without any Virtual Channels [15]. DOR has a disadvantage of routing traffic through hotspots, which increases congestions and latency

Turn based routing algorithms prohibits a certain turn (e.g. West to North) which efficiently removes any chance of a livelock while providing a better balance of traffic in the NoC. Alternative turns must be completed if the desired turn is prohibited. Traffic would not be routed in a straight line in turn based algorithms leading to a longer route to the destination. Odd Even (OE) routing algorithm provides deadlock and livelock avoidance and avoids routing all the traffic in one dimension. OE distributes traffic more evenly than DOR algorithms which lower the average latency in the NoC [3]. OE routing also prohibits east to north and east to south turns on even columns and north to west and south to west turns on old columns.

2.6.4 Adaptive Routing

An adaptive routing algorithm relies on a set of variable information to make a decision on the route selection. Adaptive routing discussed in this thesis will make decisions based on the congestion information to improve latency of a packet between the source and destination core. By balancing the NoC traffic and rerouting packets to other parts of the network using congestion information will avoid hotspots and congestion regions in the NoC, which improves throughput as well as reduces the average latency for the entire network.

2.6.5 Hybrid Deterministic and Adaptive Routing

DyAD (Dynamic Adaptive and Deterministic) routing developed by Hu and Marculescu [25] switches between adaptive and deterministic depending on the traffic load. It is one of the few examples of both types of routing used in a NoC. DyAD suggests that adaptive routing will only occur when congestion occurs within the NoC. Hybrid deterministic and adaptive routing algorithms have not been explicitly applied in recent works such as DyXY [24] or DBAR [5]. The selection of the more optimal direction only occurs when a certain traffic load occurs at the router hence they act similar to a hybrid algorithm. Therefore, most routing algorithms implementations are either deterministic or adaptive as there is no need to explicitly monitor the congestion load to control when adaptive routing should occur.

2.7 Congestion Awareness

Congestion awareness can be catorgorized in three types, locally adaptive, regionally adaptive and globally adaptive. The amount of congestion data received by each router depends on the type of congestion awareness chosen. Locally adaptive does not need traffic flow between routers as the amount of credits available from the credit value registers (CVRs) is sufficient while globally adaptive need a large amount of data transfer to hold all the congestion values to each downstream router. Congestion awareness generally sacrifices data transfer to determine a better path for a packet to route to its destination.

2.7.1 Locally Adaptive

Research on adaptive algorithms began with locally adaptive routing as shown in early implementations such as DyAD [25] and DyXY [24].These implementations rely solely on congestion data of its neighbours to select the next hop direction. Preferred Output Adaptive Routing [26] and DyXY have used the available credits as the congestion value to adaptively determine the direction for the next hop. DyAD uses an external 1-bit signal to indicate if there is any congestion or not. Due to the fact that locally adaptive algorithms are greedy in nature, the output port selected may not be the best direction. Hu and Marculescu proved [25] that DOR (XY) routing performed better than their DyAD routing for uniform traffic. This is also proven by Gratz et al. that locally adaptive routing performs worse than RCA due to its greediness [4]. Another technique, Neighbour-on-Path (NoP) utilizes a similar neighbouring monitor like DyXY [27]. NoP allows non-minimal path routing to avoid the congested NoC area. Although this improves latency, its non-minimal routing can lower the injection rate before its saturation.

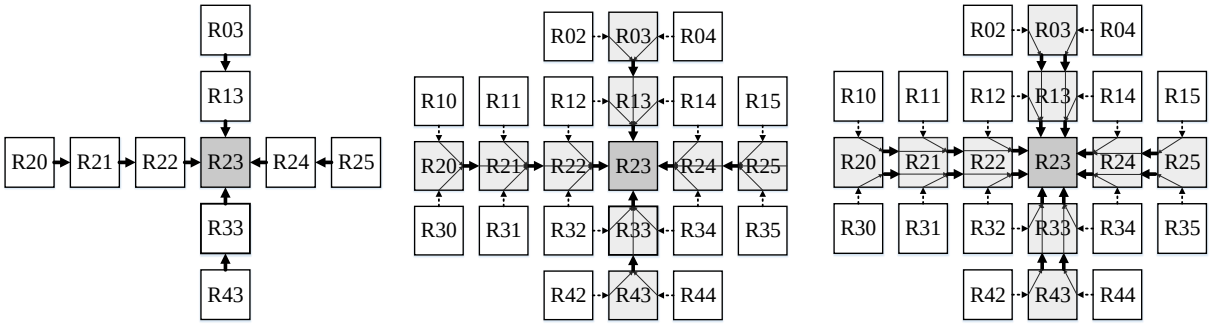
DyAD switches between adaptive and deterministic routing depending on traffic conditions. Hu and Marculescu have chosen 60% as the threshold to switch between deterministic to adaptive routing. DyAD uses the OE routing algorithm, which prohibits certain turns depending on the current location of the packet, and when there is a choice to select one of the two output direction, DyAD will select the output port with no congestion. When both outputs are not congested or both are congested, then the algorithm chooses one of them randomly.

Preferred Output Adaptive Routing and DyXY are similar and would choose the direction with more credits available when choices are available. Preferred output routing makes use of route look-ahead while DyXY does not.

2.7.2 Regionally Adaptive

Regionally adaptive routing relies on the congestion information in the neighbouring routers as well as the information of the intermediate region provided by the neighbouring routers. Gratz et al. have implement and demonstrated that Regional Congestion Awareness (RCA) produced better results than locally adaptive routing [4]. RCA has been implemented using a sideband network to propagate the congestion data. Congestion data is aggregated with the local congestion information (credits available) and then propagated by combining information of one to three directions depending on the RCA implementation. Weights are assigned to the local and non-local congestion data in the aggregation process to control which set of data have a greater effect on the adaptive routing mechanism. RCA is implemented in three different methods, RCA 1D, RCA Fan-in and RCA Quadrant. RCA 1D only aggregates

information, RCA Fanin aggregates information from the row or column as well as the neighbours of the row or column's routers. In RCA Quadrant, congestion data is only aggregated in two directions leading to slightly better result but with the use of twice as much wiring overhead.



(a) RCA 1D **(b) RCA Fan-in** **(c) RCA Quadrant**

Figure 2-6 - Congestion traffic information flow in Regional Congestion Awareness

Figure 2.6 shows the congestion traffic flow of RCA through a sideband network. Each direction requires a link for downstream router to the source router in the centre of the diagrams in Figure 2.6. Each router in the diagram have another link for transmitting congestion data in the opposite direction. RCA 1D and Fan-in requires eight additional links while RCA Quadrant requires 16 additional links. Each router propagates its congestion data to its neighbour at every cycle. Figures 2.7 shows the RCA Router Microarchitecture with the congestion awareness component shaded in gray. Please note that the additional communicate links are needed for RCA.

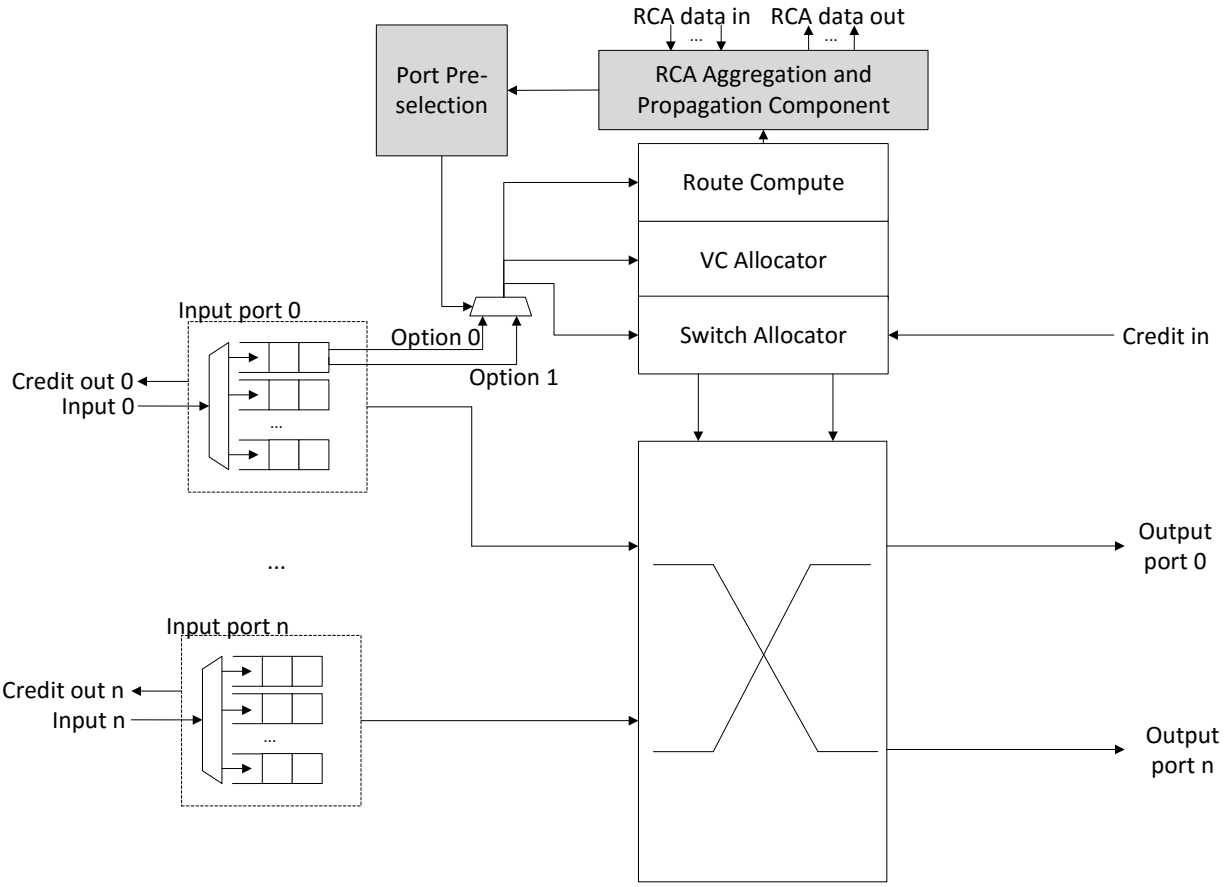


Figure 2-7 – RCA NoC Router Microarchitecture

2.7.3 Destination Based Adaptive Routing Algorithms

Regional adaptive routing has proven to be a useful methodology for improving performance as demonstrated by the technique of RCA [4]. Although RCA uses neighbouring congestion data to improve the decision making for route computation, it uses general congestion data within the region instead of data specific to the destination. RCA will deliver data beyond the destination to the upstream routers which can lead to incorrect routing decisions, e.g. congestion occurs beyond the destination. Gratz et al. demonstrated a breakthrough with RCA but it suffers from interference by data aggregation and propagation processes [4]. The

congestion data received simply is not the raw data but instead data may contain unnecessary noise. Newer implementations of regionally adaptive routing do not aggregate the congestion data. Destination based adaptive routing employs more accurate congestion awareness information by utilizing the congestion data from the source to the destination router. This technique is employed in DAR [5] and DBAR [6] which lead to more accurate routing decisions and improved performances over general regional congestion awareness data used in RCA.

Destination-based Adaptive Routing

Destination-based Adaptive Routing (DAR) uses latency as a measurement of congestion and it broadcasts the pure data to neighbouring routers [5]. DAR uses a ratio to split the incoming traffic to either the X or Y direction to balance the traffic load of the downstream routers. The ratio varies depending on the congestion conditions. Figure 2.8 illustrates how the congestion data flows in the region to propagate from the source router e.g. R30. Routers shaded in black already have the congestion data from R30. Routers in gray are currently processing the data from router R30 while white routers have yet to receive the data from router R30. At time zero, R30 propagates data to the adjacent routers R20 and R31. At time 1, R20 will propagate to routers R10 and R21 while R31 will propagate to R21 and R30. Router R21 would receive latency values from both X-direction (from R20) and Y-direction (R31). R21 would use latency info from both directions to determine a split ratio favouring the least congested direction. At time 2, data continues to propagate to Routers R00, R11, R22 and R33. For accuracy and removal of stale data, latency values would only propagate within a 7x7 frame of the NoC. For routers that would receive latency values from two routers, the lower value is propagated.

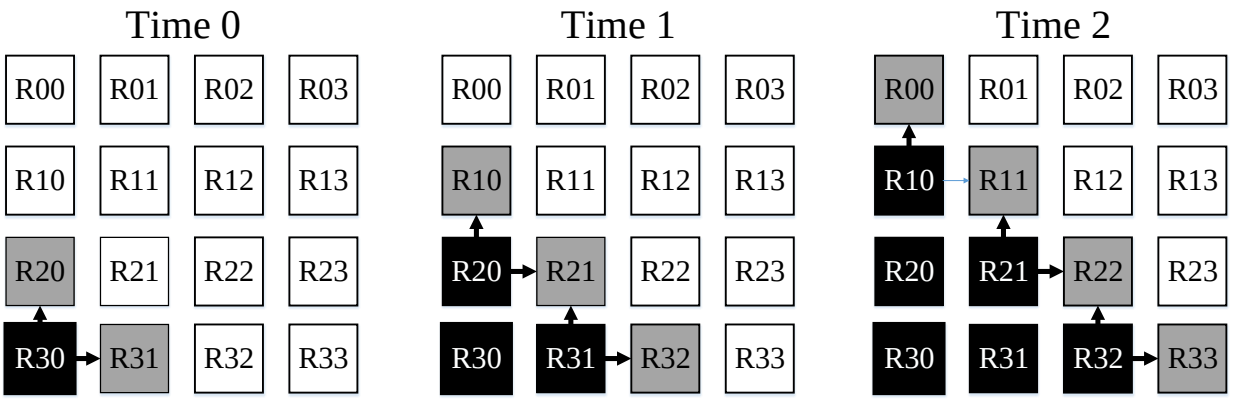


Figure 2-8 - Congestion traffic information flow in DAR

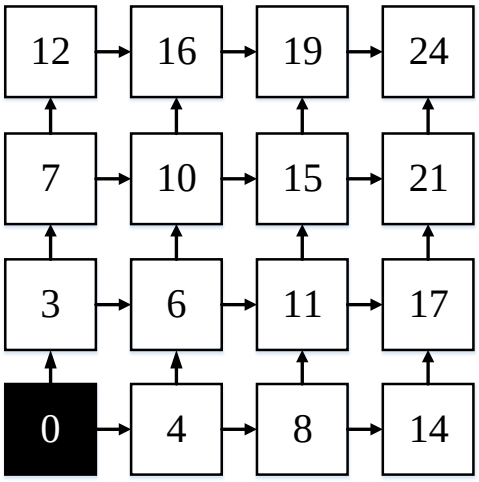


Figure 2-9 – DAR Congestion (Latency) to All Nodes

Figure 2.9 shows an example of a congestion (latency to each node) array with the value of latency stored for all routers. Please note that the bottom left router is the current router in the example. For a small 4x4 NoC example, 8 bits is enough storage for latency. For larger mesh topology NoCs such as 16x16 NoC, the router would need to use 10 to 12 bit registers to store larger latency values.

Destination Based Selection Strategy

Destination Based Selection Strategy (DBSS) for Adaptive Routing (DBAR) broadcasts one bit of traffic information to its neighbours similar to DAR [6]. Both DAR and DBAR routes the packet through the preferred output port based on the destination. DAR and DBAR both estimate the shortest path to the destination that differs from RCA, which does not take destination into account. Figure 2.10 shows two different path that the current node (Black) to take to reach the destination (Gray). DAR estimates the latency between the two paths while DBAR routes the packet depending on the congestion conditions. Note that both DAR and DBAR behave in the same if the current node and the destination are in the same dimension.

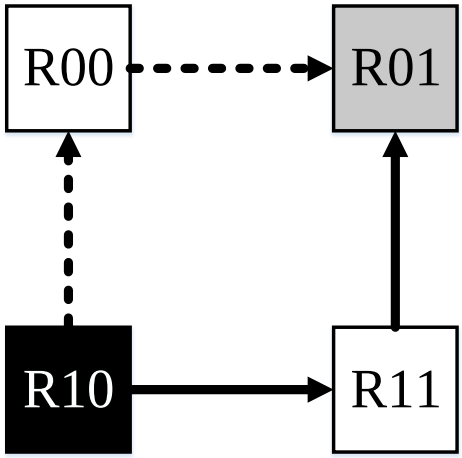


Figure 2-10 - Two Different Paths for Destination Based Adaptive Routing

The difference in congestion data receiving procedures between DAR and DBAR is illustrated in Figures 2.8 and 2.11. Figure 2.9 showed that DAR propagated congestion information to all the neighbours while DBAR only propagates congestion data for up to eight routers in the same dimension (for 8x8 mesh). In Figure 2.11, DBAR could only receive information from one dimension. Routers R10 would receive information from R00, R20 and R30 from the X-direction while receiving R11, R12 and R13 in the Y-direction. Since it is not

known if there exists a lot of congestion, this greedy routing method hopes that there is less congestion along the path.

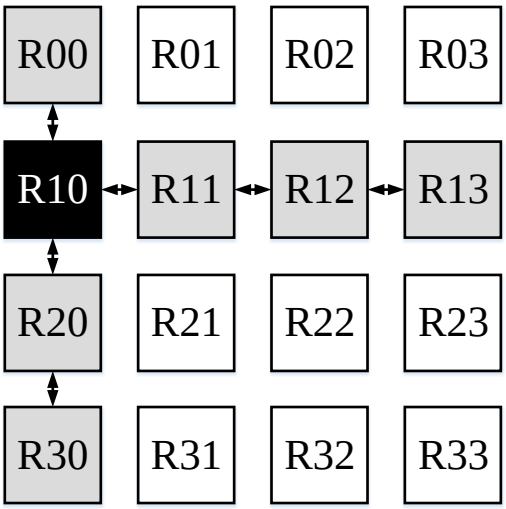


Figure 2-11 - Congestion Traffic Flow in DBAR

Similar to RCA, a sideband network is employed for both destinations based adaptive routing DAR and DBAR. Destination based adaptive routing estimates the congestion from all the routers to all other routers.

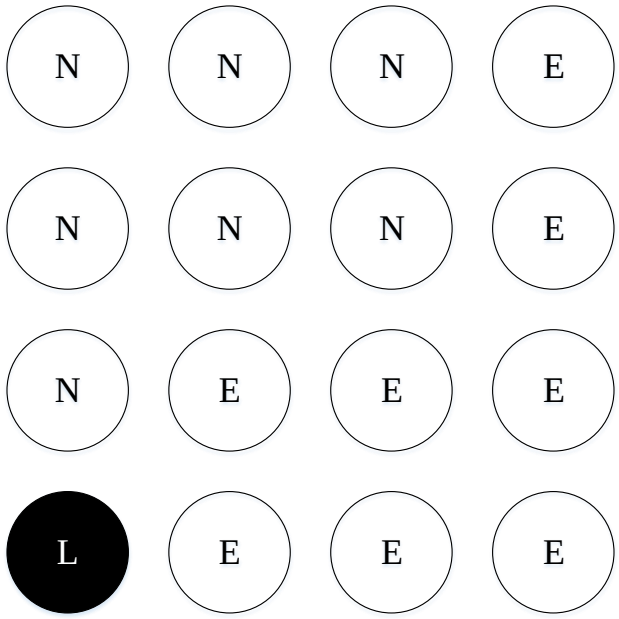


Figure 2-12 - DBAR Preferred Directions Map Example

In Figure 2.12, an example of the preferred output map is shown in an array. This map is updated every clock cycle and will change with respect to congestion in the NoC. This is implemented as a lookup table for packets. A preferred output will be returned when a header flit requests for its desired output port with the packet's destination ID.

In the case of DBAR, an array of registers will store the preferred directions to the destination. During the direction selection process, the direction indicated by the preferred direction array would be used. Each register will just need 3 bits to store the five directions (four cardinal plus local).

In terms of hardware complexity and overhead, DBAR is much simpler than the complex latency computation modules in DAR. Both employ external links leading to the wiring overhead. When compared with RCA 1D and Fan-in (8bits), DBAR requires wider links (9bits) to implement the sideband congestion monitoring network. DBAR computes latency in three

cycles allowing the links to be reduced from 15bits to 5bits. RCA 1D, RCA Fanin, DAR and DBAR all need 8 links to send and receive the congestion data.

Other Attempts

Farahnakian et al. used a table and also embedded a few bits in the header to store congestion information [28]. This method uses a table to keep track of alternative routes for comparison with the standard route. If the alternative route is found to have a lower latency between source and destination, the path would be chosen based on this information. Farahnakian et al. only tested this method on a small (4x4) mesh NoC. A larger NoC would lead to lower performance and an exponentially larger table.

2.7.4 Globally Adaptive

If there is a way to have the knowledge of the complete network, routing will become much easiler. Instead of relying on greedy algorithms with locally and regionally adaptive routing strategies, globally adaptive routing can avoid the mistake of routing to a hotspot and instead bypass a congested region completely.

In the works of Tedesco et al., a message is routed through the network while the first packet will establish a path [29]. Subsequent packets in the message will follow the same path until congestion is discovered and a new path is established. Tedesco et al. suggested that fixed path routing is better than regionally adaptive routing such as RCA. Each node along the path would monitor congestion and send it back to the source router allowing the source to have information of the entire path to have a global view. The congestion information is sent back via

a congestion packet within the standard NoC communication channel in place of to a sideband network. The drawback to this strategy is that a message sent is assumed to have a large number of packets. For short messages or single packet, this work has very low potential to improve the performance in a network.

Ramakrishna et al. suggests that by piggybacking congestion information in the header flit, eventually each node will have the congestion information of the entire network [13]. The globally adaptive router would first compute the shortest path to the destination based on the congestion information available on that particular router and route the packet to the next hop. The downstream router will recompute the shortest path and route the packet using its congestion data. Since each router has different congestion data and when the current node is closer to the destination, the router will have more accurate congestion information. The path will be pre-selected with the available congestion information, and the router will also benefit from port pre-selection.

2.7.5 Congestion Awareness Monitors

Yuan et al. suggested that regional congestion awareness can be improved for congested hotspot regions by using source routing and a congestion agent sending information to other routers when congestion is detected in a hot spot [30]. This method has shown improvement over RCA in congested hotspot traffic at the cost of more embedded data of each hop in the header for source routing. Moreover, this method would not show significant improvement under standard synthetic traffic.

2.7.6 Congestion Aware Scheduling

While congestion awareness is employed to improve routing, others have attempted to shape traffic and improve scheduling to avoid congestion. Chao et el. have suggested that a reconfigurable scheme along with a congestion aware scheduling algorithm can reduce congestion for repeated traffic patterns [31]. Another proposal is to design a congestion aware scheduler for an application specific NoC, especially for AMBA bus [32]. These scheduling algorithms are not as useful for general cases.

2.7.7 Summary of Methods

Table 2-1 – Summaries of Methods

Method	Congestion Information Transferring Method	Congestion Data Transfer Rate between two routers per clock cycle	Monitoring Region	Data Accuracy
Locally Adaptive	Local data only	N/A	Neighbouring routers (up to 4 routers)	Neighbouring: high No data beyond neighbouring routers
Regional Congestion Awareness (RCA)	Sideband network	RCA-Single or RCA-Fan-in: 8+8 bits RCA-Quadrant: 16+16 bits	8x8 Region with accurate data within a few hops	Neighbouring: high Regional: low
Destination Based Adaptive Routing (DAR)	Sideband network	5+5 bits	7x7 Region	Moderate due to slow update rate
Destination Based Selection Strategy for Adaptive Routing (DBAR)	Sideband network	Combined 9 bits for both uplink and downlink	All routers inline with the x and y axis	High
Global Adaptive Routing (GCA)	Embedded in Header Flit	Varies (based on header flit frequency)	8x8 Region	Moderate to high depending on update rate

2.8 Summary

In this chapter, the concept of NoCs is described as well as congestion awareness. Different methods of routing, simulation and adaptive routing are discussed to illustrate the number of varieties available in NoC. NoC topology is described as well as data transmission. The router architectures for different adaptive routing implementations are introduced along with speculation and look-ahead routing to reduce latency for each packet transmission through the routers. Different flow control methods are introduced and how they affect congestion awareness data. Adaptive routing and congestion awareness is also discussed and explained in this chapter.

Chapter 3

CAAR:

Congestion Aware Adaptive Routing

Congestion awareness allows a router to monitor its surrounding and mode better decisions to be made to improve data transfer latency within a NoC. Basic congestion awareness techniques allow the monitoring neighbouring routers while more complex strategies let the router monitor the entire surrounding region and even the entire on-chip network. Recently, congestion awareness information has been used for mainly adaptive routing. In this chapter, congestion awareness for other NoC routing decisions such arbitration will be explored. Furthermore, the development of a congestion aware adaptive routeing (CAAR) will be explained.

3.1 Congestion Awareness for Allocators

NoC router allocation can be customized to improve performance for all packets. By default, round-robin allocation scheme would give equal share of time for all the packets that would impact packets travelling far distances in a NoC. There have been proposals to improve

allocation for the source by creating a job scheduled to ensure certain packets are transmitted on time [33]. Such techniques would prioritize packets based on the tasks that need to be completed. Congestion Awareness for allocation would improve latency for prioritized tasks. Congestion affects each flit of these packets as they compete for arbitration with the closer from near routers flits. Improving allocation by allowing long distance packets to have priority under congested conditions could possibly improve overall latency within the NoC. Since prioritizing packets would create additional latency for other packets, prioritization should only be used under congestion conditions. There are two approaches to altering arbitration by firstly changing how VCs are allocated with the VC allocator and secondly how flits are allocated by the switch allocator.

For our implementation of CAAR, we have chosen to use regional congestion awareness (RCA) as the baseline regional adaptive router since it offers significant improvement over local adaptive routing by employing a moderate hardware. Destination based adaptive routing methods such as DAR and DBAR requires more complex hardware that does not improve NoC throughput significantly. RCA is also better for low congestion data transfers and is more suitable for embedding congestion data into the header.

3.2 VC Allocator

VC allocation is an important process where a newly arrived packet arrivals are assigned a VCID in the downstream router. It is critical that fairness, deadlock and livelock avoidance are considered during this process. In the case of multiple packets requests for an output port, an arbitration scheme is necessary to determine the packet that would be granted the output port first.

Normally, an arbitration scheme that is selected should ensure fairness that guarantee to avoid starvation for certain packets. Traditionally, Round-Robin scheme is employed to ensure that these conditions are met. Round-Robin is an excellent arbitration scheme under normal traffic conditions.

In the proposed method, Round-Robin is modified to ensure no packets are starved while improving the throughput of the NoC. Some packets will suffer higher latency. If some packets that suffer higher latency due to the large amount of hops from source to destination are prioritized, the overall average latency can be improved.

With destination-based adaptive routing such as DAR [7] and DBAR [8], and approximate estimate of congestion is measured for the current source to destination. Using the congestion data, the modified round-robin procedure can deny any VC allocation requests if its path to the destination is greater than a specified threshold. If traffic congestion persists for that packet, it is important to avoid starvation. A register is used to track the number of cycles that the packet is spent waiting. When the counter reaches a maximum allowed waiting time, the packet will be treated as an uncongested packet and would be allocated normally by round-robin mechanaism.

3.3 Switch Allocator

Switch allocation is a process where flits are selected for crossbar transversal at every cycle. Under normal circumstances, Round-Robin will accept only one VC request at a time and then prioritize for the next VC request. Once a packet allocates a flit, it will have to wait up to a full loop for all the other VCs to be allocated. To reduce average latency and improve packets that travel through a long distance, round robin is modified to allow those packets to allow two flits to

be allocated in two consecutive cycles before allowing the round robin counter to increment for the next VC.

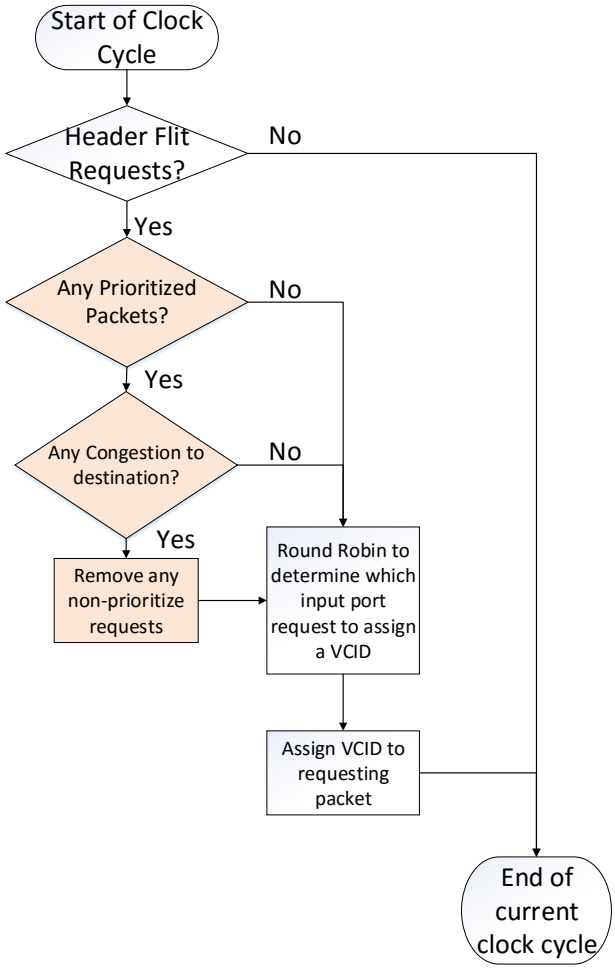


Figure 3-1- Congestion-Aware VC Allocation (CAAR-VA) Flow Chart

Packets that have a prioritization flag will indicate to the routes that it could be prioritized. It is important to limit the number of packets being prioritized to prevent non-prioritized packets from suffering from starvation. Normally for standard mesh-based NoC routers, only packets travelling through a long distance (i.e. packets travelling from one corner to the other corner of the NoC) should be prioritized. Of course, CAAR would be extended to QoS applications where certain packets are given priority to ensure a lower latency.

Our proposed VC Allocation Procedure: Figure 3.1 illustrates its flow chart to determine if CAAR will prioritize any VC allocation requests. When an unassigned packet requests for an output VCID and is prioritized, CAAR would ignore non-prioritized requests and allocation between based on congestion prioritized requests only.

Our CAAR architecture includes congestion awareness based prioritization with both the VC and switch arbitration. CAAR-VA prioritizes VC allocator while CAAR-SA only prioritizes the switch allocator. Similarly, Figure 3.2 shows the flow chart for prioritizing the flits at the switch allocator.

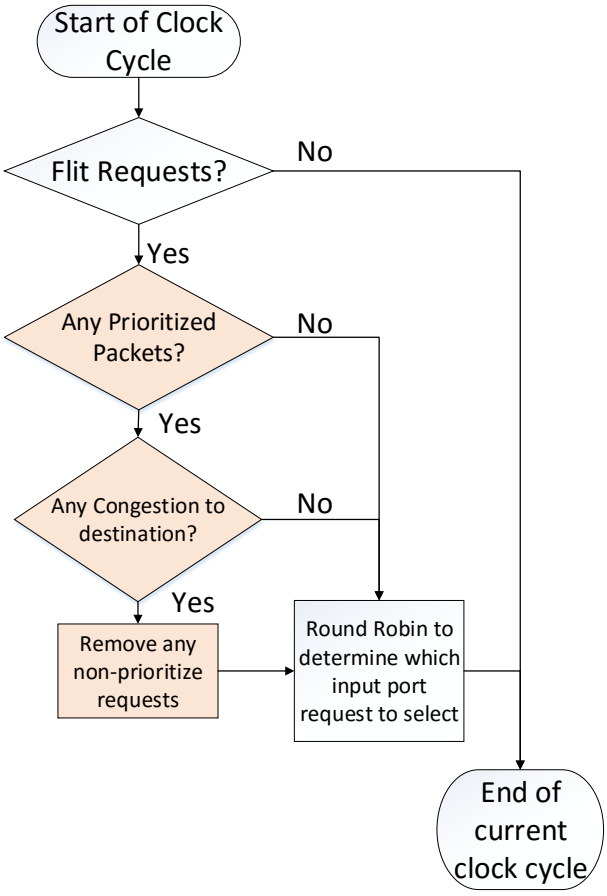


Figure 3-2 - Congestion-Aware Switch Allocation (CAAR-SA) Flow Chart

3.4 Prioritizing Packets

Packets can be prioritized to reduce latency of reaching from source to destination at the expense of higher latency for other packets. Packet prioritization could achieve a number of objectives. One is to improve the average latency of the overall NoC communication and the second is to guarantee throughput for Quality of Service (QoS) applications. Das et al. has presented a method of prioritizing the packets based on application awareness under heavy loading to improve latency within the NoC [34].

3.4.1 Long Distance Packets

In a mesh topology NoC, traffic imbalance is created by the placement of routers in a 2D grid. Any data transmissions in the central part of the mesh will experience less congestion than to data transfers between corners to a 2D mesh. In a situation where the centre of the mesh has more activity, the packets being sent from one corner to the other corner of the NoC will suffer higher latency. In this situation, many packets can be transmitted between two central nodes while one packet transmission has yet to complete from corner to corner. To combat this situation and to improve the throughput and latency of these packets, they can be prioritized when congestion arises. These packets will have better latency and the VCs occupied can be reused by the other packets.

The source router is able to calculate and identify how far a packet will travel in the NoC. In our CAAR router, we reserve 1 bit in the header flit. This allows the packet to be labeled as prioritized under congested conditions.

3.4.2 Quality of Service (QoS)

Elementary QoS enabled NoCs are separate packets into two categories, best efforts (BE) and guarantee throughput (GT) as proposed in Æthereal, which is one of the original NoC architectures that supports packet prioritization for QoS [35]. Packets that need the lowest latency would be prioritized and labeled as GT packets. Past work had focus on separating VCs into two separate classes for BE and GT packets. This limits the efficiency of VC usage effectively. When no packets need to be prioritized, GT designated VCs are left unused leading to undesired utilization of buffering spaces. Attempts are made to prioritize packets by shaping traffic to improve bandwidth allocation [36]. Unfortunately this method also increased hardware usage by 200%. Wang and Bagherzadeh suggested that local adaptive routing and a uniform buffering space for both BE and GT packets support QoS which would improve latency and performance [11].

In our CAAR router, QoS can be enabled by allowing source cores to enable prioritization for the desired packets. Round Robin arbitration is modified to service prioritized packets first allowing prioritization under congested networks. Unlike Wang and Bagherzadeh's method [11], we propose in CAAR that congestion triggers prioritization. In case where two packets from two VCs from the same input port are prioritized, the packet that is heading towards a congested direction in the NoC would receive priority.

3.5 Hardware Implementation of CAAR

3.5.1 Hardware Implementation of Requestor

It is necessary to identify a few parameters for every input request to the allocator to understand if it would generate a request or not. In order to determine which request would be prioritized, one has to determine two factors:

For each input port, which VCs have a prioritized request?

Among all the VCs of all input ports, does every output direction have at least one prioritized requests?

The second factor is required which there is contention between two or more requests originating from different input ports.

Figure 3.3 illustrates the proposed modified requester to the allocators for the switch allocator. In a simple requestor, a three-input AND gate is used to determine that a request to the allocator exists along with a signal indicating that the input VC is assigned then the credit is available in the output port of the packet requesting from the input VC. The requester circuit is made of combinational logic that allows it to update with new congestion data before the next clock cycle. This allows the allocator to the service prioritized packets as congestion arises.

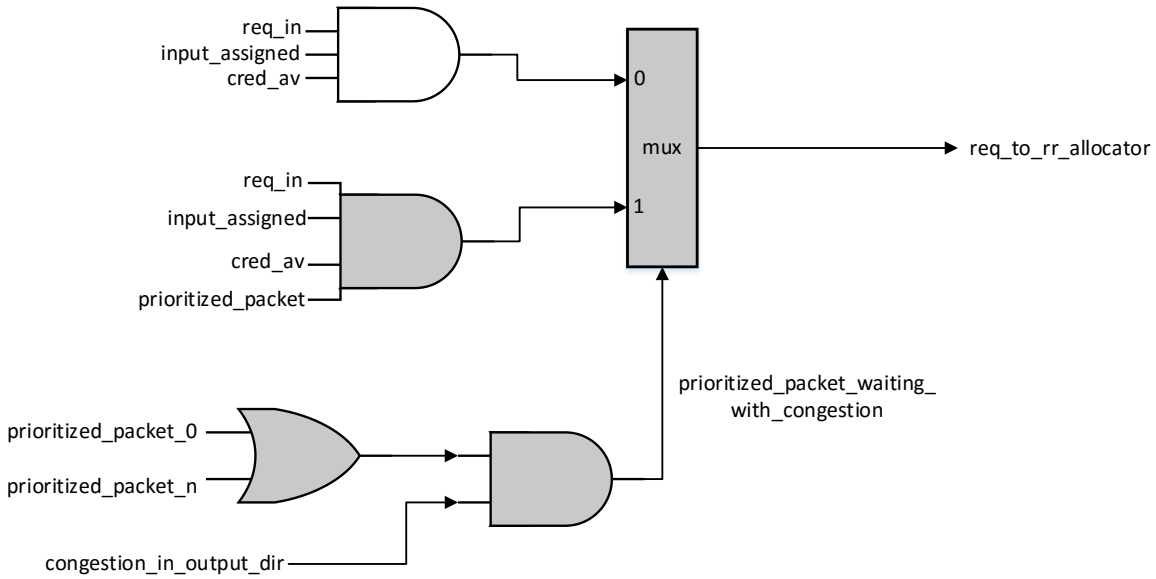


Figure 3-3 – A Congestion Aware Prioritization Switch Allocator Requester

We have introduced a few logic gates to check if congestion exists and which input VC has to be prioritized. When congestion exists, the mux would select the second AND gate. This AND gate will be set only if this VC request needs to be prioritized. By setting all the multiplexors for each VC in one input port effectively, only the VC with a prioritized request will be granted.

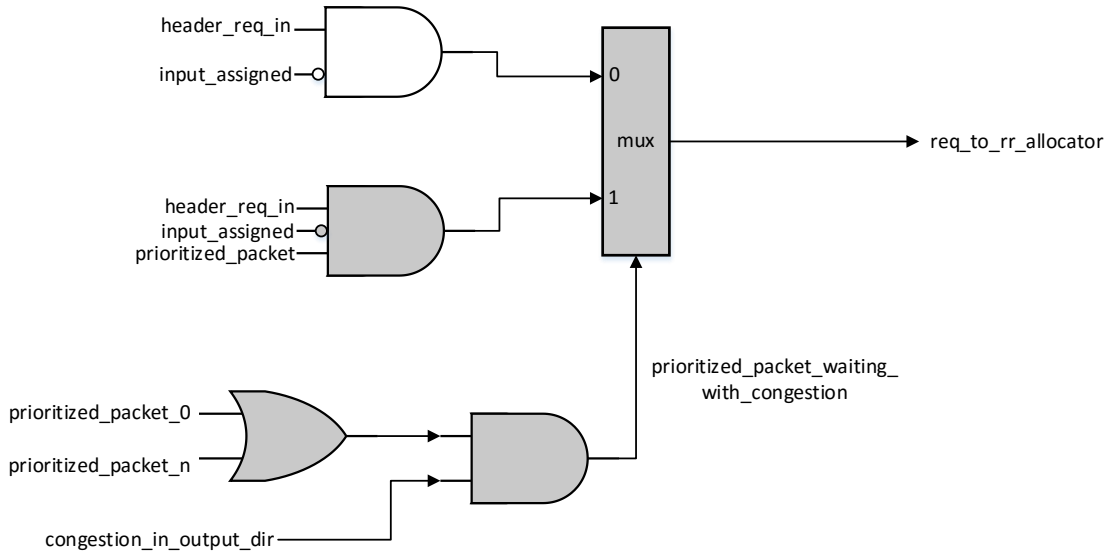


Figure 3-4 - A Congestion Aware Prioritization VC Allocator Requester

Similarly, Figure 3.4 presents the design requester circuit for the VC Allocator. This would allow the VC allocator to prioritize, which input VC will be assigned to an output VCID first. This will ensure that if there are a few packets waiting to be assigned an output VCID, the prioritized packet will be granted first.

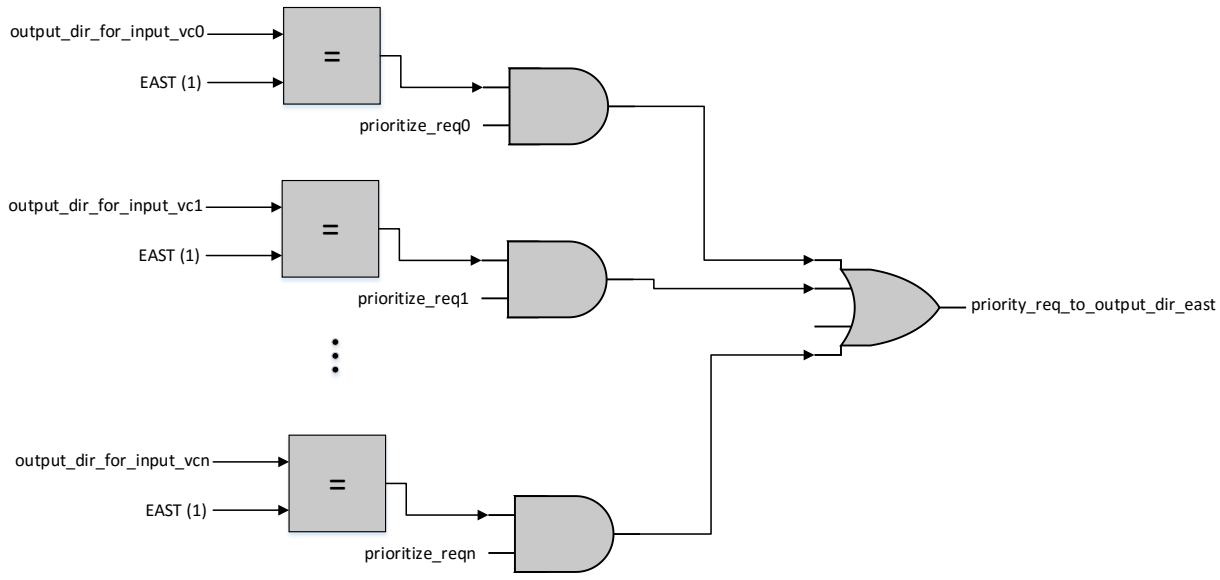


Figure 3-5 - Circuit to Determine Prioritize Request to Output Direction

3.5.2 Allocator Modification for Congestion Data

Out router implementation will modify the standard iSLIP allocator to accept congestion data. The simplest modification is to reduce the number of requests to the RR allocator at a particular clock cycle.

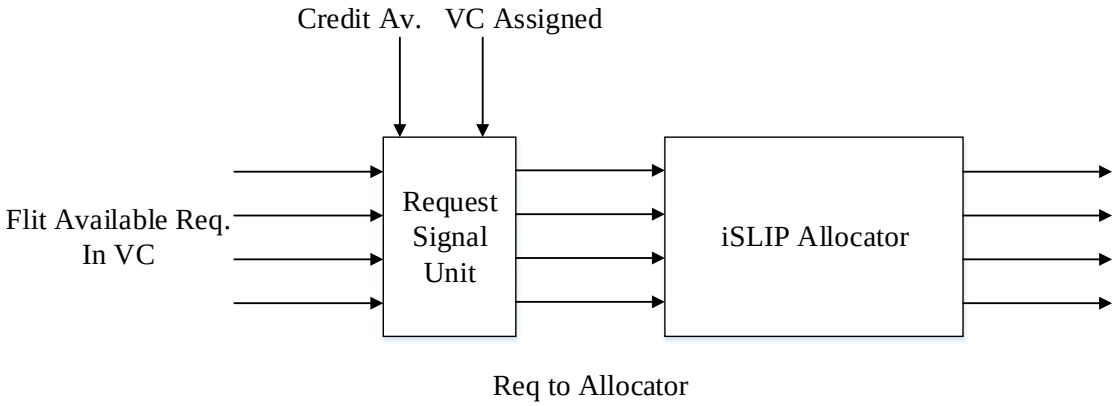


Figure 3-6 – Hardware for Request to iSLIP RR allocator

Figure 3.6 shows how the request system operates with the RR allocator would normally work without any congestion data. The input port VC requests needs to be check if the output VCID is assigned and there is credits available in the downstream router before the request would be granted to the RR allocators.

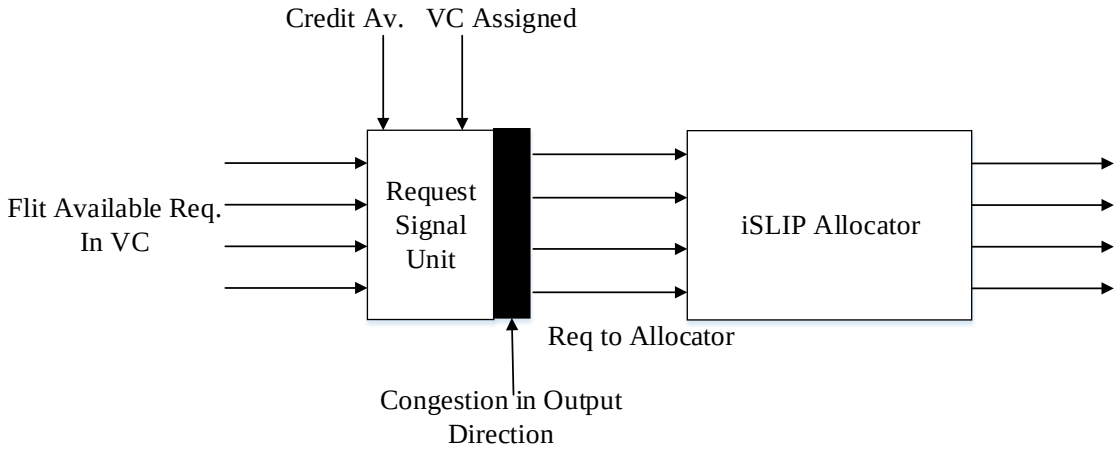


Figure 3-7 – Additional Hardware for Request to iSLIP RR allocator with Congestion Prioritization

By simply adding an addition circuitry to the request system, congestion data can be used to control which VC will gain access to be allocated first. A packet with congestion in the output direction can be assigned a VC faster than other VCs, which can possibly reduce overall latency. The same can be employed for the switch allocator.

3.5.3 Hardware Implementation of VC Buffers

NoC router flit storage buffers exist in different forms including simple FIFOs, SRAM buffer and dynamic buffers [14]. We have simulated and modelled a standard sized SRAM buffer is commonly used in many NoC research experiments.

The buffer models of a SRAM type buffer that has two pointers (read and write) to indicate where the buffer's next read and write location. A simple subtractor will be able to determine when the buffer is full. Figure 3.8 shows the implementation of the standard SRAM buffer mode. Each flit arriving at the buffer is analyzed for its VCID and then stored in the

corresponding VC selected by the input mux. The buffer control unit hosts the read and write pointer along with the subtractor hardware to generate the request signal for any flits available to be processed in the VC buffer.

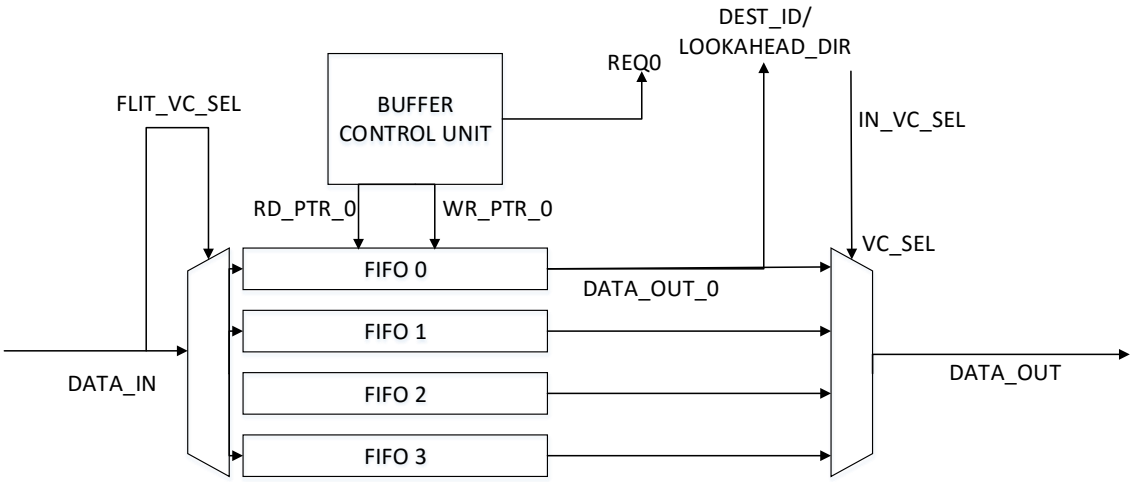


Figure 3-8 – Standard Size SRAM buffer

Another type of buffer that is modeled is the dynamic sized buffer. The concept of a dynamic sized buffer is relativity new compared to standard sized buffers. The dynamic sized buffer is implemented by a large size of SRAM for all the VCs of one input port along with the individual read and writes pointers for each VCs. An address list of free locations is available for newly arrival flits to be allocated in the SRAM. Once the flit is allocated to the SRAM, another list tracks all the addresses of flits allocated to that specific VC as shown in Figure 3.9. There is a separated list of addresses for each VC.

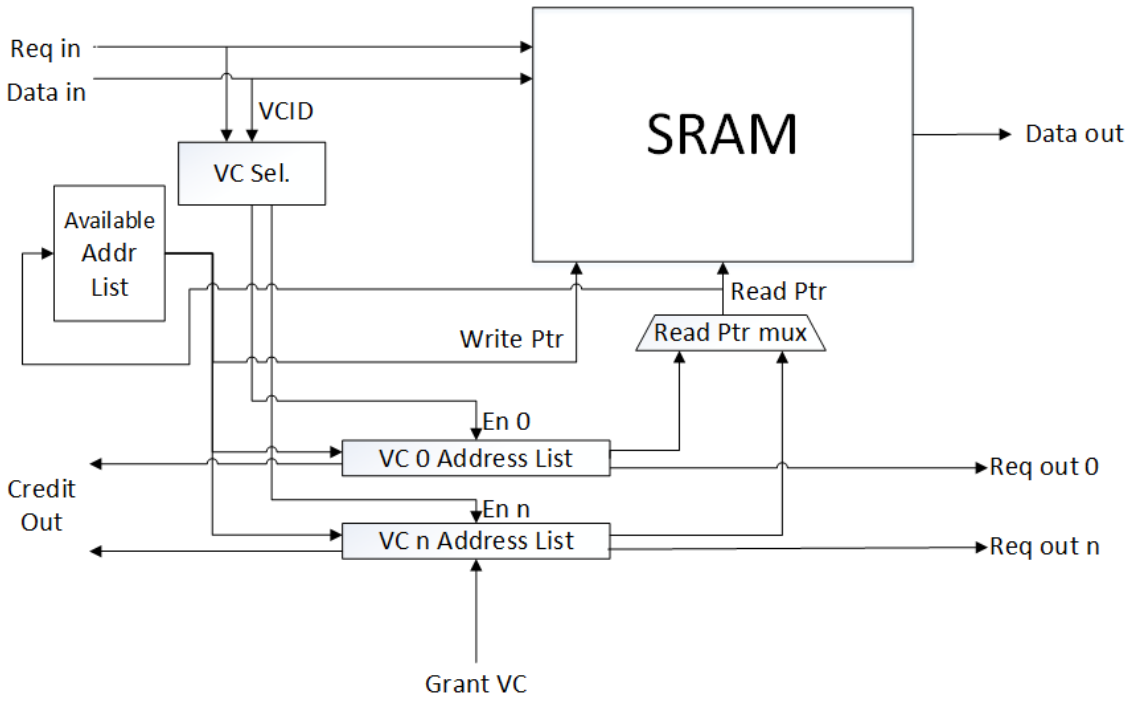


Figure 3-9 - Dynamically Sized Buffer

3.5.4 Look-ahead Routing Unit

A latency reducing method is to hide the latency of the route computation by determining the output direction in advance in the upstream router in parallel to the VC and switch allocation process and embed the output direction(s) in the header flit of the packet. Since the output port of the packet is already known, that router ID is also known that allow the next hop to be determined. Also this allows the route computation to be able to be completed in the upstream router.

The purpose of the look-ahead routing unit is to complete the routing computation task in advance in the upstream router. This unit is very similar to the route computation unit but instead

of using the current router's router ID for route computation, it will use the downstream router's router I.

Figure 3.10 shows the architecture of a look-ahead routing unit. The next hop ID composes of a few adders and subtractors that will determine the next hop router ID. This figure shows two directions generated (options 0 and 1) for adaptive routing. Deterministic routing would just have one direction. The directions would be stored in the header flit and would be read by the downstream router.

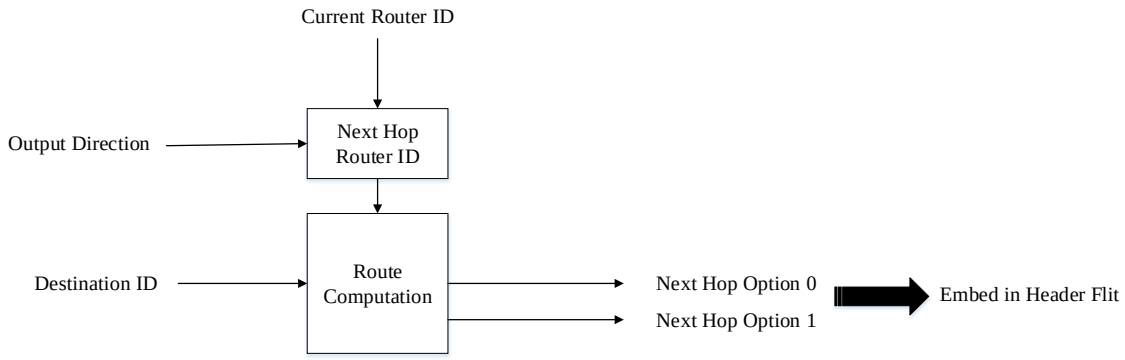


Figure 3-10 - Architecture for Look-Ahead Routing

3.5.5 CAAR Congestion Information Propagation

In CAAR, we propose to embed congestion information into the header of every packet. This allows reduction of links and the elimination of the sideband network which would reduce switching power especially under the condition when the NoC remains relatively congestion free. The negative effect is that the congestion conditions update will be slow. Other congestion awareness methods such as DAR perform well with periodic updates of three cycles **[**5**]**. A small component would monitor the input flits and extract any congestion information found in the

header. That information will be routed to the congestion awareness component which will analyze the information and update the congestion tables accordingly. When a packet is being transmitted downstream, a header modifier component will be embedding the congestion information for downstream routers.

3.6 CAAR Router Microarchitecture

Figure 3.11 illustrates the components and their interconnection required to extract and embedded congestion information into packet headers in our CAAR router micro-architecture.

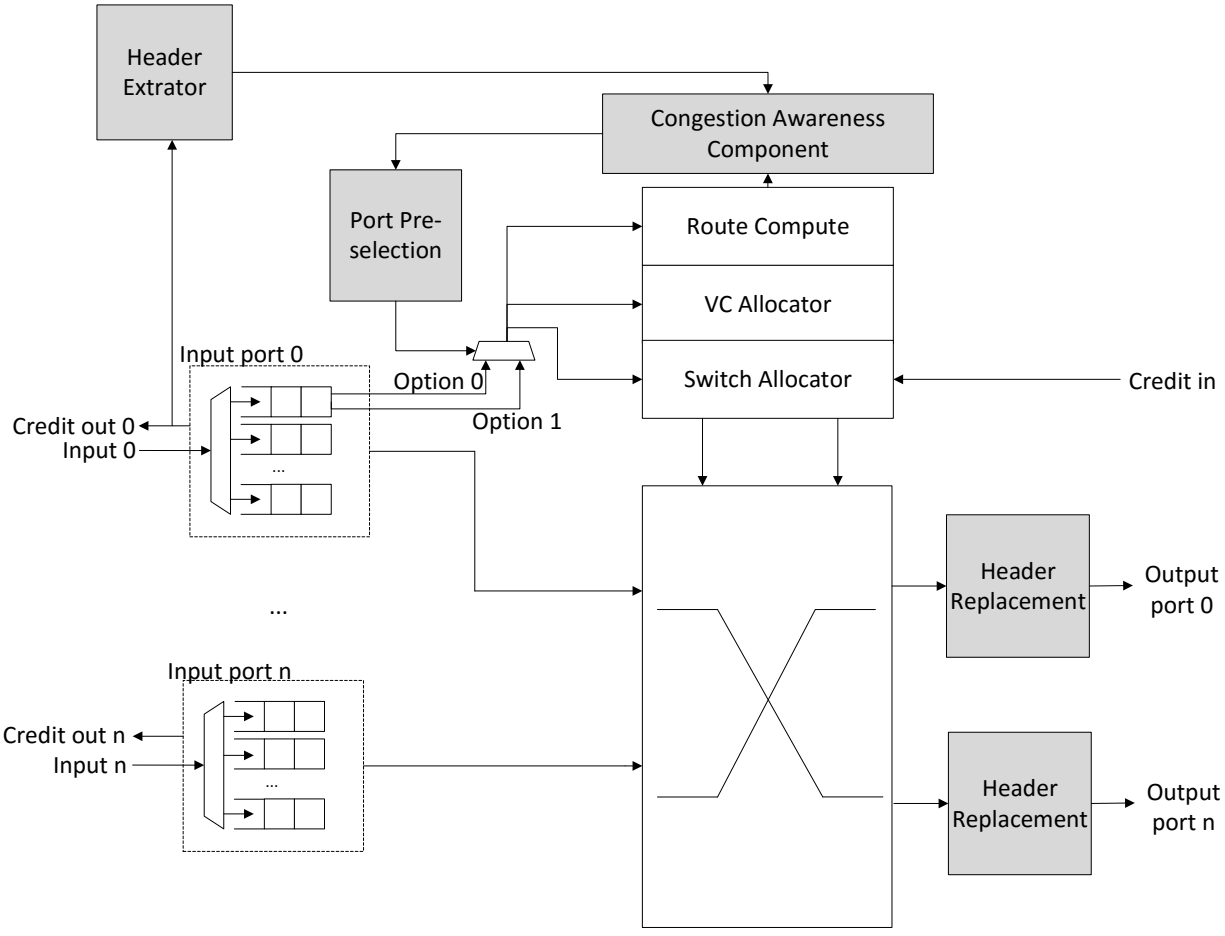


Figure 3-11 – CAAR Congestion Awareness Router Microarchitecture

The CAAR router includes the standard input port buffers for storing the flits that will be sent to a downstream router. There is a multiplexor at the output of every VC buffer to determine the output port direction (i.e. X or Y direction). The port pre-selection will select option 0 for the X direction and similarly option 1 for the Y direction. The port pre-selection hardware unit uses data from the congestion awareness unit to determine the output port resulting in a lower latency. The arbiter hardware that consists of the route computation unit, the VC allocator and the switch allocator unit will process information for the flits in the buffer and determine their next hop direction and allocate them to the output port. The additional details on the NoC router pipelining will be provided in section 4.5.

3.7 Summary

This chapter explored the methodology and design for key components to construct CAAR. This chapter gives details behind the methodology for the implementation of CAAR along with how CAAR will utilize congestion awareness information to improve allocation to provide better throughput for long distance packets. This chapter also describes how CAAR can be extended to QoS application with packet prioritization. Finally, this hardware design behind the packet prioritization is illustrated.

Chapter 4

CAAR Modeling and Simulation

This chapter outlines the experimental setup and process for testing and researching the congestion aware NoCs being investigated in this thesis. In this chapter, the development of a suitable simulator is discussed as well as testing methods. The chapter describes the development of each simulator components as well as traffic generation of each packets and hardware modeling of each unit. We also discuss how to generate different types of traffic pattern and testing features in the simulator for debugging and experimental testing.

4.1 SystemC Simulation

SystemC is a powerful extension to C++ programming language allowing hardware to be modeled in software. C++ already supports object orientated programming that is necessary for creating multiple instants of each component [37]. In our case, it would be the NoC router.

A NoC network simulator is developed using systemC to model a transaction accurate router in an expandable mesh topology NoC. The router models a pipelined architecture with the hardware descriptive components including the buffers, arbiter, crossbar and any additional

congestion awareness hardware. The simulator is able to accept different parameters under different setups such as comparison including dimension order routing (DOR), locally adaptive routing (e.g. DyXY) and regional adaptive routing (e.g. RCA, DAR). The number of virtual channels (VCs) and size of the buffer for each VCs are adjustable. Moreover, the traffic generation is adjustable so that we can compare its performance under different traffic patterns related to varying NoC applications.

4.2 Hardware Modeling

The router components are designed and modeled separately before it is added to the overall NoC. SystemC supports object oriented programming making it easier to create multiple instances of each hardware components for reusability. The NoC system simulation will track all the packets as they are injected into the NoC by assigning a corresponding packet ID. This would allow the NoC to verify that all the packets reach to their correct destinations as well as the packets being stuck in the NoC when a livelock or deadlock situation occurs.

Figure 4.1 shows the process of developing the NoC model for the simulator to test and investigate our CAAR based methodology. First of all, individual components such as buffers, arbiter and the crossbar are developed. The second step is to combine all these components to create the CAAR based NoC router for a mesh topology. Finally, multiple routers are combined together with physical channels to create the mesh topology based NoC.

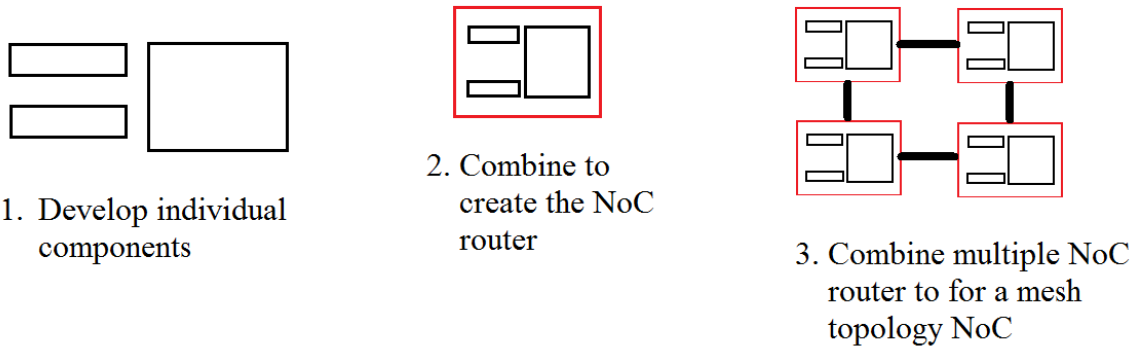


Figure 4-1 - Building a NoC by Components

4.3 Router Modeling

The NoC router is built from three main components, the buffers, arbiter and the crossbar. Figure 4.2 shows a simplified architecture of a NoC router. Please note that the number of buffers in a NoC router is determined by the number of input ports of the router. For a mesh NoC, there are five directions, the four cardinal directions and one local direction for the core. In the simulator, each one of these components are developed separately and then combined to create the NoC router. Once that router is verified to work correctly, it can be used to build a mesh topology NoC.

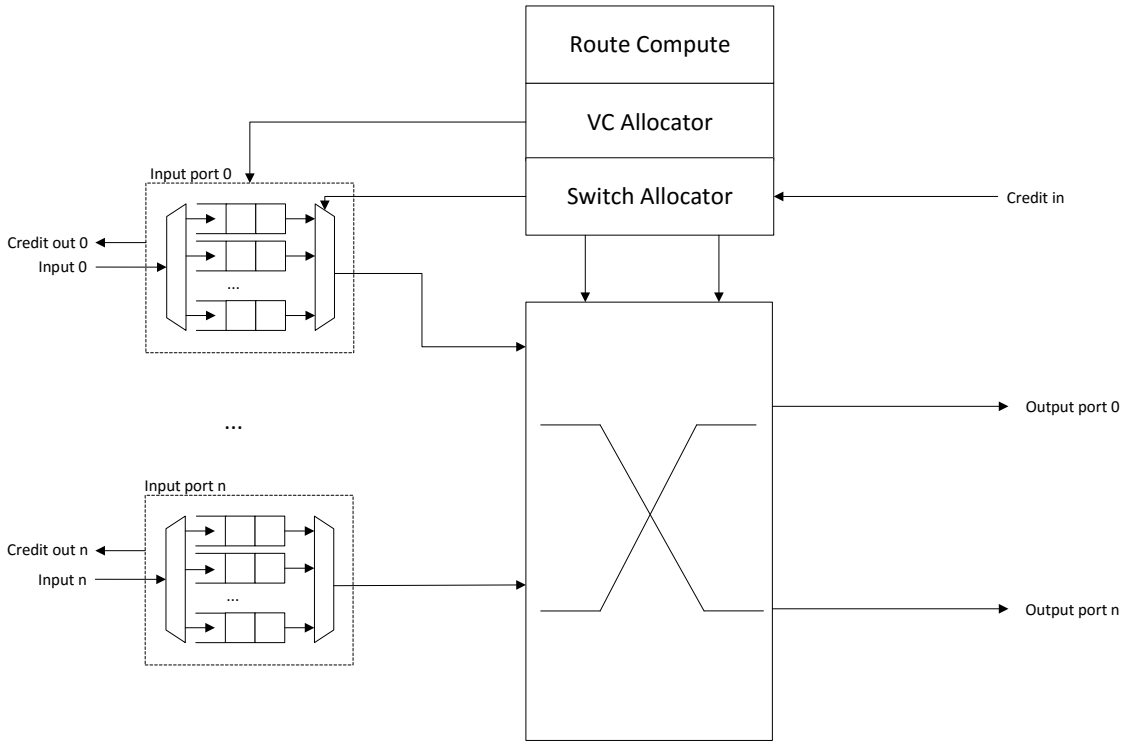


Figure 4-2 – Baseline NoC Router Architecture

4.3.1 Buffer

Figures 4.3 and 4.4 provide the pseudo-code for writing and reading a flit in the router buffer respectively. The write process would only occurred if the enable signal is set by a request in the input-port and a proper VC is selected by the specific VCID. Figure 4.5 shows the read pointer increment code, once the arbiter grants the flit for transversal through the crossbar. The buffer capacity is only for five flits per buffer, and three bits of data will exceed the number of flit slots available. Therefore, it is necessary to check and reset the pointers when necessary.

Buffer Write Process:

```
for (i=0; i<NVC; ++i) {
        if (en[i].read()) {
        // increment the write pointer for current vcid
                if (wr_ptr[i].read() < FIX_BUFFER_RAM_DEPTH-1) {
                        wr_ptr[i].write(wr_ptr[i]+1);
                } else {
                        wr_ptr[i].write(0); // if max value, reset
                }
        }
}
```

Figure 4-3 - Buffer Write Process

Buffer Read Process:

```
for (i=0; i<NVC; ++i) {
        if (req_inc_vc_ptr[i].read()) {
        // increment the read pointer for current vcid
                if (rd_ptr[i].read() < FIX_BUFFER_RAM_DEPTH-1) {
                        rd_ptr[i].write(rd_ptr[i].read()+1);
                } else {
                        rd_ptr[i].write(0); // if max value, reset
                }
        }
}
```

Figure 4-4 - Buffer Read Process

4.3.2 Dynamic Sized Buffer

Figure 4.5 shows an example code for finding a free flit slot in the SRAM buffer. The process is implemented as combinational logic with logic gates and multiplexors. The SRAM pointer is the write pointer for the next available flit from the input port to be written to the buffer.

The read and write processes are similar to the standard SRAM buffer. The only difference is that the addresses are updated with pointers instead of being provided from a list.

Next Flit Slot Section Process:

```
if (flit_slot_valid_table[0].read()) {
        flit_slot_avaliable.write(true);
        _sram_wr_ptr.write(0);
} else if (flit_slot_valid_table[1].read()) {
        flit_slot_avaliable.write(true);
        _sram_wr_ptr.write(1);
...
```

Figure 4-5 - Free Flit Selection Process

4.3.3 Arbiter Components

The arbiter components are the most important hardware units in the NoC router. These components make all the decisions in the router. The arbiter is mainly responsible for routing, VC allocation as well as switch allocation. Regional congestion awareness and adaptive routing also introduce some additional components in the arbiter to detect and quantify congestion and make more complicated routing decisions.

4.3.4 Route Computation

The route computation (RC) unit has the purpose of identifying the output port direction for all the incoming packets. This unit is built by using combinational circuit that allows fast switching instead of using a sequential machine that would compute the results in a full clock cycle. The addition of look-ahead routing will allow route computation to simply look at the next hop direction stored in the embedded data in the header flit of a packet. Since adaptive routing can have up to two output port directions available, both directions (if both directions exist) are

necessary to be stored in the header flit. The congestion awareness unit will provide the optimal directions for each quadrant. The simulator simply reads the data in the current flit.

4.3.5 Allocators

VC-router based allocators usually have two staged, one for the input VC contentions and the second for output port contentions. There are many different ways to implement the allocators such as a simple round robin (RR), a modified Round-Robin scheme known as iSLIP [23], or a more advanced method known as matrix allocation. In comparison to other simulators developed for congestion awareness research, iSLIP allocation has been employed as it is considered to improve the allocation throughput and easier to implement as compared to matrix allocators. Our simulation environment employs the modified RR allocator i.e. iSLIP for both VC and switch allocators.

The iSLIP allocator is a two stage allocator that can be divided into three parts, the requester, granter and accepter. The requester determines the input direction that will trigger a request to the output direction. If there are multiple input directions requesting for the same output direction, the granter will perform RR to determine, which input direction will be granted. The accepter will accept one of the input VCs for allocation when there are multiple VCs requesting for the same output direction.

Figure 4.6 shows the architecture of an iSLIP, RR allocator used in our CAAR simulator for both the VC and switch allocator. The allocator is modeled by utilizing only combinational logic, which is fast and would allow other tasks such as assigning the VCs and storing the values to registers to take place after the allocation.

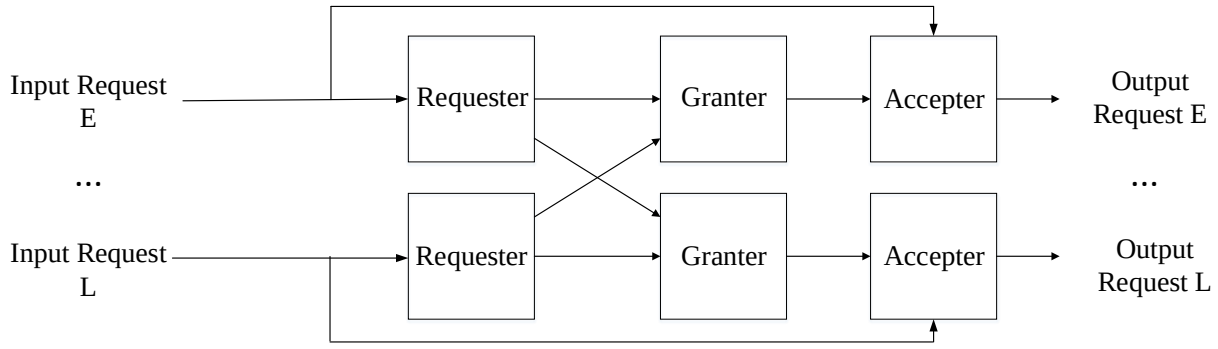


Figure 4-6 - Architecture of the iSLIP Two-Staged RR Allocator

Figure 4.7 shows how the simulator checks all the input VCs of an input port that is assigned to the output port 0 and determines if any of the VCs have a request to that output port.

iSLIP Requester Process:

```
void rr_requester::assign_request_0 () {
        bool temp_req = false;
        for (j=0; j<NVC; ++j) {
                if (req_in[j].read() && output_dir[j].read() == 0)
                        temp_req=true;
        }
        req_to_output_dir[0].write(temp_req);
}
```

Figure 4-7 - iSLIP Requester Process

The simulator determines which input direction will be granted the next request to be assigned a VC or crossbar transversal. Figure 4.8 shows how the simulator selects an input port with switch and if statements.

iSLIP Grant Process:

```
switch (grant_ptr.read()) {
        case 0:
                if (req_in[0].read()) {
                        grant_input_dir[0].write(true);
                        grant_input_dir[1].write(false);
                        grant_input_dir[2].write(false);
                        grant_input_dir[3].write(false);
                        grant_input_dir[4].write(false);
                        granted.write(true);
                        granted_dir.write(0);
                } else if (req_in[1].read()) {
                        grant_input_dir[0].write(false);
                        grant_input_dir[1].write(true);
                        grant_input_dir[2].write(false);
                        grant_input_dir[3].write(false);
                        grant_input_dir[4].write(false);
                        granted.write(true);
                        granted_dir.write(1);
                }
.....

                } else {
                        grant_input_dir[0].write(false);
                        grant_input_dir[1].write(false);
                        grant_input_dir[2].write(false);
                        grant_input_dir[3].write(false);
                        grant_input_dir[4].write(false);
                        granted.write(false);
                }
                break;
```

Figure 4-8 - iSLIP Grant Process

A similar process applies for the iSLIP accepter which is used to select an input port VC if there are contentions for an output port.

4.3.6 Virtual Channel Allocator

The virtual channel allocator is one of the arbiter components responsible for allocating downstream VCs for newly arriving packets. It will also release any VCs once the last flit has been allocated by the switch allocator. The modified RR allocator, iSLIP is used in the simulator to select which VC gets to be assigned an output VCID. In the case of a contention between two inputs VCs to be allocate to the same output direction, only one of those VCs will be assigned an output VCID.

The process to assign an output VCID is given below:

i. For all the output ports, determine if any VCs are free and select (if available) which VCID will be assigned next.
ii. For all the VCs, is there a header flit of a packet that is requesting for an output VCID?
iii. For the packets that require an output VCID of the desired output port, is a VC free and available to be assigned?
iv. If an output port VC is available, RR arbitration will determine which input VC will be assigned an output VCID in case of any contentions.
v. The input and output VCIDs are recorded in a table. The corresponding output port is set to occupied and cannot be used by another packet unit it is released by the current packet.

4.3.7 Switch Allocator

The switch allocator assigns the flits to an output-port direction. If there is any contention, by default RR will decide which flit will occupy the output port for that cycle. The switch allocator is implemented with the modified RR allocator, iSLIP for maximum throughput.

The switch allocation process occurs as follows:

i. For all the VCs, is the packet assigned a downstream VCID?
ii. For each of the requesting VCs, are there any credits available in the output port?
iii. Are there any contentions among any of the output ports?
iv. If not, select the VCs that can be allocated. If there is any contention, RR determines which input-port is selected.
v. If multiple VCs from one input-port are selected, RR determines which input VC is allowed to be allocated.

4.4 Congestion Detection and Control Modeling

Congestion detection and control is enabled by the additional hardware components that monitor the congestion conditions in the surrounding routers. Additional links may be used to improve the transfer of congestion information between routers.

The simulator would use three different congestion measuring methods to determine any congestion in the simulated NoC. The three methods are availability of buffer spaces in the downstream router, free VCs availability in the downstream router and crossbar contention. The first method will count the number of free buffer space available in the downstream router. Fortunately with a credit based flow control, the router itself knows how many free buffer spaces are available without the needs to receive any external congestion information from the neighbouring routers.

Figure 4.9 illustrates how the local router can calculate the amount of buffers space available in the downstream router by simply adding all the credits in all the VCs of an output port. A port pre-selection unit will determine the direction to be selected for each quadrant.

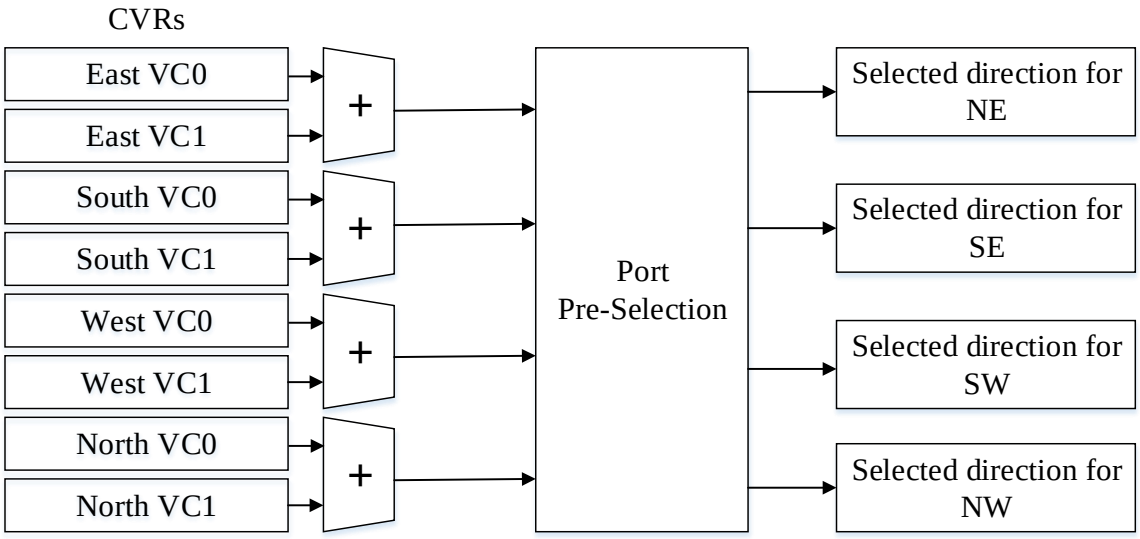


Figure 4-9 - Locally Adaptive Buffer Availability Computation

Similarly the number of free VCs available or crossbar contentions can be computed using a similar architecture as shown in Figure 4.9.

4.4.1 Locally Adaptive Routing

Locally adaptive routing for congestion control allows the route computation unit to select between the X or Y directions for any packet if the destination is diagonally located from the current router. A congestion awareness adaptive routing methodology, DyXY requires four

registers that tracks the direction to be selected for all the four quadrants that will be updated at each cycle. Depending on the congestion around its neighbouring routers, it will store either the X or Y direction. All locally adaptive routing algorithms for the mesh NoC are based on the process similar to DyXY.

Figure 4.10 illustrates the additional hardware such as multiplexors employed for locally adaptive routing. Since the simulator uses look-ahead routing to improve latency, the two directions are already computed in the upstream router and appear as two options embedded in the header flit. A small hardware unit will check the congestion condition of all the directions and store the less congested direction in every clock cycle. Figure 4.10 shows that one of the two options is selected for each VC.

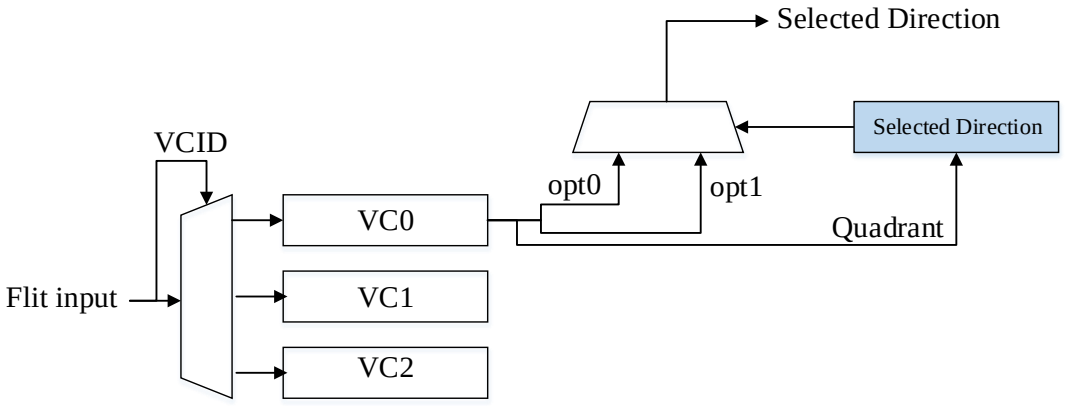


Figure 4-10 - Locally Adaptive Hardware

Locally adaptive congestion awareness algorithms would read register values on the current router to determine which directions would be selected for each quadrant. Using a method given in sections 4.4 (describe in the previous page), congestion values can be obtained.

4.4.2 Regional Congestion Awareness (RCA)

Regional Congestion Awareness (RCA) developed by Gratz et al. [4] was one of the first NoC router microarchitecture that took regional congestion data into consideration for adaptive routing. Our developed SystemC simulator implements this method by utilizing additional links. Figure 4.11 illustrates how a sideband network co-operates with the standard mesh NoC.

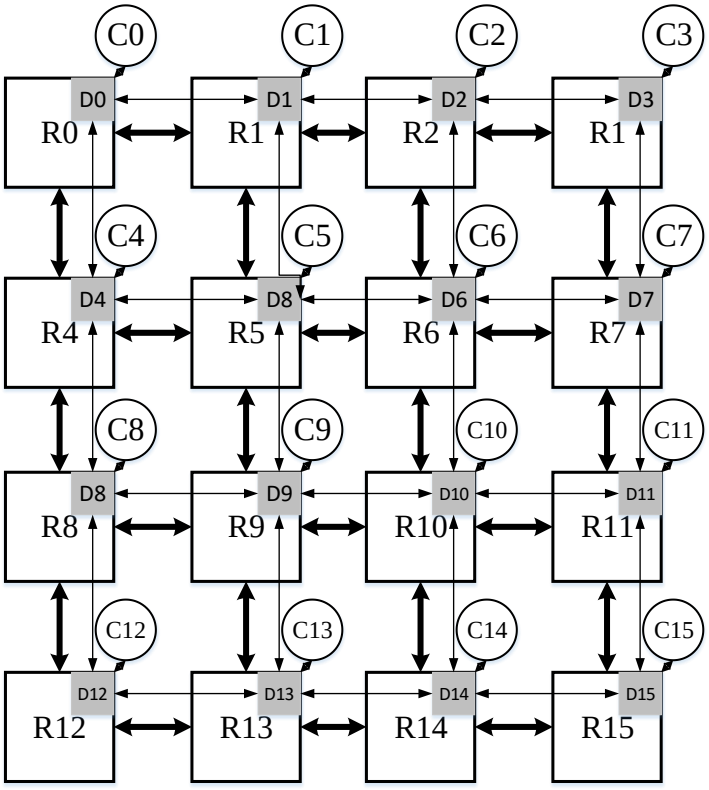


Figure 4-11 - Regional Adaptive Routing with Congestion Information Links

The congestion data links are modeled similar to the physical channels for flit transmission. Instead of a flit data structure link, a congestion information data structure is used to define the links in the SystemC based model. This allows the congestion data to be dynamically expandable in size allowing various types of congestion data required for testing.

RCA Data Propagate Process:

```
void xvc_router::process_propagate_rca_data() {
        rca_data_out[EAST].write((rca_data_in[WEST].read()+_rca_used_vc_in_output_dir[WEST].read())/2);
        rca_data_out[WEST].write((rca_data_in[EAST].read()+_rca_used_vc_in_output_dir[EAST].read())/2);
        rca_data_out[NORTH].write((rca_data_in[SOUTH].read()+_rca_used_vc_in_output_dir[SOUTH].read())/2);
        rca_data_out[SOUTH].write((rca_data_in[NORTH].read()+_rca_used_vc_in_output_dir[NORTH].read())/2);
}
```

Figure 4-12 – RCA Data Propagate Process

Figure 4.12 provides the SystemC code used to implement the RCA based data propagation process. Each direction would receive data every cycle. The data is processed and combined with local data before it is transmitted to the next router. All the RCA processing are completed independently from the normal NoC router operations.

4.4.3 Destination Based Adaptive Routing

Destination Based Adaptive Routing (DBAR) requires a sideband network similar to RCA's sideband network shown in Figure 4.11. In DBAR, the sideband network bus size varies between the routers depending on its location in the mesh NoC but the total bus lines between the two routers are nine lines. There are also two additional buses connecting the routers to transmit the destination ID of any next flits being transferred.

Figure 4.13 provides the pseudo code for the implementation of propagation data between two routers. First of all, it is necessary to determine the number of VCs assigned. This would be computed with combinational logic hardware. Please note that when some congestion exists, a value of 64 is added because it's is the weight of most significant bit of a 7-bit value. This indicates that congestion at the local router is the most important and the same value is propagated to the next router.

DBAR Data Propagation Process:

```
dbss_tmp_cnt=0;
for (j=0; j<NVC; ++j) {
        if (_input_vc_assigned[EAST][j].read())
                ++_dbss_tmp_cnt;
}
tmp_v=dbss_data_in[WEST].read();
tmp_v=tmp_v>>1;
if (dbss_tmp_cnt>4)
        tmp_v+=64;
dbss_data_out[i].write(tmp_v);
```

Figure 4-13– DBAR Data Propagate Process

Destination based adaptive routing means that each router calculates the best routing path to the destination router. This requires that each direction is to be calculated for every router. A comparison between congestion values on both the X and Y axis are required and the determined value must be stored so that it can be used by the route computation process. For simulation, a simple 2D array Boolean variable is employed where 0 represents X-direction and 1 means Y-direction.

4.5 Pipeline Modeling

A high performance NoC router employs pipelining to improve the operating clock frequency by separating the routing tasks into different stages. Normally, a NoC router is separated into three stages. The SystemC based simulator we developed is able to model all the stages at transaction level that is very accurately modeled as compared to hardware.

The different stages for a packet to transverse the VC wormhole router include: Buffer Write (BW), Route Computation (RC), VC Allocation (VA), Switch allocation (SA) and Crossbar Transversal (XT). Link Transversal (LT) or Cross Channel is not a stage within the NoC router as it does not require any clock cycle to propagate the flit from one router to another.

The pipelining model of NoC routing is achieved by modeling each hardware component separately at the transaction level. Although it takes longer to simulate, the simulation process is comparable and similar to hardware transactions. Figure 4.14 shows a packet passing through a router's pipeline stages that is performing perfectly without any stalls. This is usually the case with the NoC that is relatively free.

	0	1	2	3	4	5	6	7
Header	BW+RC	VA+SA	XT	LT				
Body1		BW	SA	XT	LT			
Body2			BW	SA	XT	LT		
Body3				BW	SA	XT	LT	
Tail					BW	SA	XT	LT

Figure 4-14 – Pipeline without Stalls

As packet injection rises, pipeline stalls will start to occur. Figure 4.15 shows two type of stalls that can affect the performance and latency of a packet traversing the NoC router. The first type of stall occurred when there are no available VCs in the downstream router. The header flit end up spending four cycles to transverse the NoC. All the sequential flits of the same packet will experience the same delay. Figure 4.15 also shows another type of stall that occurs when there is no buffer space available in the downstream router.

	0	1	2	3	4	5	6	7	8	9
Header	BW+RC	VA FAIL	VA+SA	XT	LT			Stall due to VA fail to allocate a VC		
Body1		BW	STALL	SA	XT	LT				
Body2			BW	STALL	SA	XT	LT			
Body3				BW	NO CRED	STALL	SA	XT	LT	Stall due to no credits avaliable
Tail					BW	STALL	STALL	SA	XT	LT

Figure 4-15 – Pipeline Experiencing Stalls

4.6 Packet Modeling

A data packet is an organization of one or more flits. Our SystemC based simulator models a flit as the lowest level of data structure and the common data transferred between the routers as well as being buffered or traversing the crossbar switch in any router. A packet has a common source and destination ID. It also contains the header flit with all the routing information and body flits. The last body flit is a special type of flit known as the tail flit. In hardware, all these types of flit require two bits in each flit to indicate their types. The simulator models a flit differs than the actual hardware. First of all, flits in hardware are specifically sized (e.g. 32, 64 or 128 bit flits). The simulator model flits with a dynamic size as a data structure in software. This allows additional debugging data items such as injection time, packet ID, etc. to be embedded in each flit for analysis. Moreover, the standard parameter such as the VCID, destination ID, flit type, etc. are also embedded in this way.

4.6.1 Packet Organization

As described above, a packet is organized into many different flits. Figure 4.16 shows that a header flit has much more information embedded than the body and tail flits. In this

example, there are eight flits. The header flit will pass through every router first and then the body and tail flits will follow.

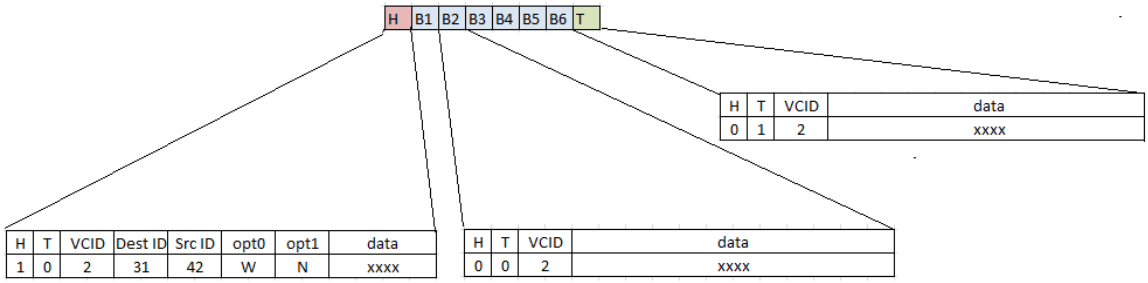


Figure 4-16 – An 8 flit Packet with Different Types of Flits

```
struct flit {
    bool data_flit;
    bool traffic_flit;
    bool header_flit;
    bool tail_flit;
    sc_uint <VC_WIDTH> vcid;
    sc_uint <LEN_X> srcidx;
    sc_uint <LEN_Y> srcidy;
    sc_uint <LEN_X> destidx;
    sc_uint <LEN_Y> destidy;

    sc_uint <DIR_WIDTH> opt0;
    sc_uint <DIR_WIDTH> opt1;

    int injection_start_time;
    int packetid;
    int flitid;

    sc_uint <NROUTER_X> east_congestion_data;
    sc_uint <NROUTER_X> west_congestion_data;
    sc_uint <NROUTER_Y> south_congestion_data;
    sc_uint <NROUTER_Y> north_congestion_data;

    bool far_distance_packet;

    sc_uint <DATA_SIZE> flit_data;
};
```

Figure 4-17– Flit Organization Structure in Simulator

Figure 4.17 illustrates the flit structure used in our SystemC simulator. This is the standard data structure for transferring and processing data within the NoC. The flit structure has a few Boolean variables to indicate what types of flit it is. The VCID is stored as well as the source and destination ID for routing and debugging. The two directions opt0 (option 0) and opt1 (option 1) are used for look-ahead routing. Several integer variables such as injection_start_time, packet_id and flit_it are used purely for debugging and testing the router and NoC model within the simulator. There is no purpose to have these fields in a hardware implementation of any NoC flits, and a simple integer type can be chosen. Standard C++ types such as integers perform faster than a SystemC types such as "sc_uint<32>" which is an unsigned integer type for hardware modeling.

4.7 Source Modeling

The source core is an important hardware unit responsible for packet generation and injection in the NoC. The source core model has a network interface to inject flits into the NoC router and a packet generator. Generated packets are separated into many flits and stored in a linked list. They are injected when the credits are available. With a linked list data structure, the simulator is able to hold unlimited flits as supported by the host computer's available memory. This allows the NoC simulator to simulate traffic with high latency without any risk of crashing the simulator.

The packet generator is required to track the packets generated in every period of 100 cycles to simulate the desire injection rate. In our NoC simulator, when a packet is ready to be injected into the NoC system, the whole packet is buffered in the source core until it can be sent. The source core acts as a packet injection controller. The source core controls the injection rate

to avoid overflowing the NoC which causes the entire mesh NoC to saturate and dramatically increase latency for all the routers.

Figure 4.18 above illustrates two example packets being generated by the source core. The packets are generated five cycles after another. Every flit of in a packet is individually generated and defined as either a header, body or tail flit. Each packet is assigned a VCID as it is required to be assigned to a VC at the source router. For debugging purposes, the packet ID, flit ID and execution time is also included.

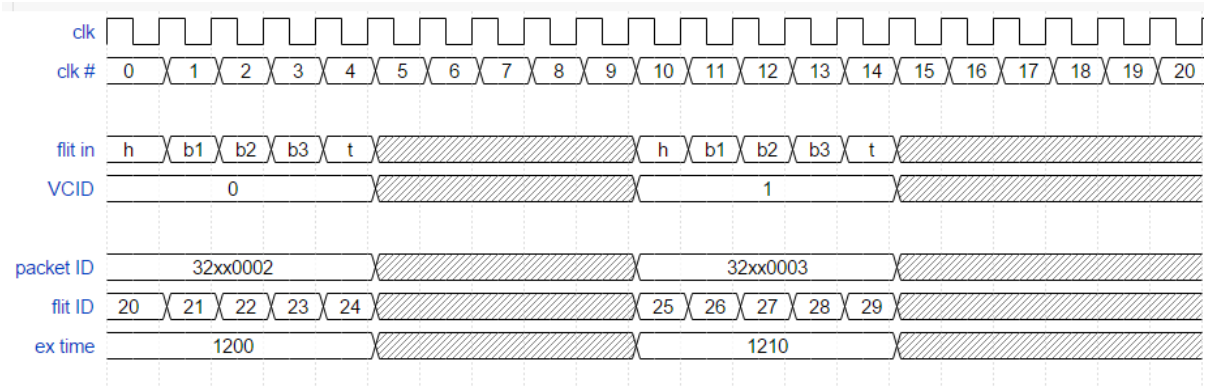


Figure 4-18 - Packet Generation Example Waveform

Figure 4.19 illustrates how a packet with four flits is injected in a NoC router. The simulator is designed using the credit based flow control. The source core tracks the number of credits available in the local NoC router and injects the packet when there are credits available meaning there is buffering space available in the VC at the downstream local router. After ‘body flit 2’, the number of credits available is zero or the downstream buffer is full, the source stops transmitting any more flits until it receives a credit_in signal to indicate a flit slot is freed in the downstream VC. This occurs when the downstream router process one of the flits in that VC.

The source will send any remaining flits at this point. The ‘tail flit’ is sent as shown in Figure 4.19. Since there are no more flits waiting to be sent after the tail flit, the source core will increase the number of credits available as it receives credit_in signals until the maximum buffer flit slots are available in the downstream router.

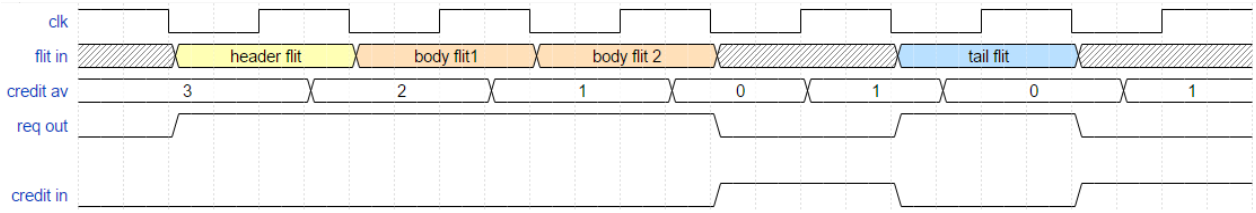


Figure 4-19 - Example Waveform of the Source Generating a Packet (4 flits)

4.8 Sink Modeling

The sink core will receive flits when they reach their destination. The sink core packages the packet together by checking their VCID. The sink core is modeled by a simple network interface to the router that models the physical channel. For synthetic traffic modeling, it is only necessary to record the arrival of all the packets and verify their VCID and packet ID. In a SystemC simulator, packet information such as injection time is embedded in each packet so that it can be used to measure latency between the source and the sink cores.

Figure 4.20 shows the waveform for receiving an example packet with four flits (one header flit, two body flits and a tail flit). The sink will read flits on the positive edge when a request is made to the sink. The sink will then process data and transmits an acknowledgement

(credit_out) to the upstream local router to release a credit. This will allow the local router to transmit another flit.

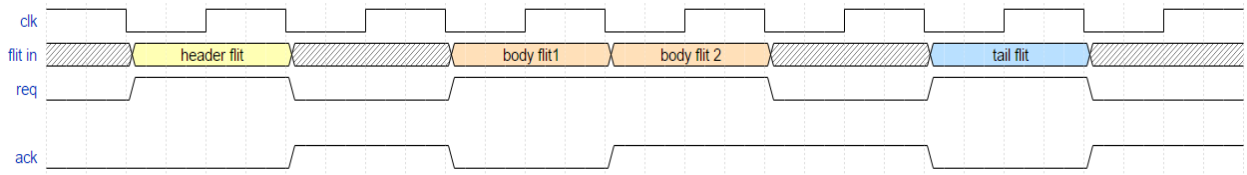


Figure 4-20 – Example Waveform of the Sink Receiving a Packet (4 flits)

4.9 Summary

This chapter reviewed the implementations of the different types of NoC architectures for testing, research and evaluation purposes. The implementation of a NoC router and each component are described in detail at the architectural level. Each component is described with their timing and transaction characteristics for hardware modeling and physical component design. In this chapter, we discussed the implementation of the transaction accurate NoC microarchitecture is as well as the development of testing components. We also discussed how the source and sink cores will handle packets under various traffic conditions.

Chapter 5

Experimental Results

This chapter will outline the experimental setup and configuration used to obtain the experimental results. The parameters for the experimentation along with the methods to calculate the performance will be described in details. Packet generation to be tested for various traffic patterns is described. This chapter will also illustrate the performance of our CAAR based adaptive NoC routing and a numbers of result graphs showing the latency for various different traffic patterns are presented. Moreover, this chapter also describes how packets are tracked to ensure correct delivery and how latency measurements are embedded in the simulation traffic packets.

5.1 Traffic Generation

NoCs are tested under various traffic conditions to determine its performance. Different traffic conditions affect the operation and congestion within the NoC. Traffic has several conditions such as injection rate, injection behaviour and traffic patterns that would affect the NoC performance. Generally the higher the injection rate, the higher the latency of all the packets to transverse the NoC. Traffic behaviours also affect congestion level for a constant

injection rate. For example, traffic that resembles a uniform process will perform differently than a self-similar process [13]. The traffic pattern of the data flow within the NoC will affect the overall performance. For testing and evaluation purposes, a specific traffic pattern can be defined to observe the performance of NoCs. Otherwise, an application can define a specific communication pattern or even completely irregular traffic in nature.

The source would generate packets and flits based on what is desired for the testing parameters. Different traffic pattern would require a slightly different definition of the destination ID for each packet. A desired injection rate is selected and the flits are injected by the source cores. We have already explained the detail in chapter 4 about source modeling.

There are several types of synthetic traffic data types. Our simulation will model the random values in terms of uniform traffic with the rand() function to generate a random value. Other traffic patterns such as transpose are generated by calculating what the destination ID would be based on in the current router.

5.2 Traffic Pattern

Traffic patterns within the NoC can be regular (synthetic) or irregular (e.g. some application related traffic). In simulation, traffic patterns are defined in the initialization phase of the simulation and the source cores will generate the appropriate traffic pattern by computing the destination ID for each packet at each router. The destination ID would then be embedded into each packet allowing all the routers on its path to read it and make its own routing decisions.

5.2.1 Synthetic Traffic Patterns

NoC performance estimation and evaluation can be measured by injected a testing traffic into the NoC with a specific traffic pattern. Some traffic patterns only allow packets to be sent to the nearby routers while in some other patterns, the packets are intended to travel across the NoC. The size of the NoC will also limit the performance of the NoC regardless of the traffic pattern under evaluation. A list of some synthetic traffic pattern for a mesh topology NoC are listed below. We will consider (x, y) to be the coordinates of each router in the NoC and N to be the size of an NxN mesh topology NoC.

5.2.2 List of Synthetic Traffic Patterns:

Uniform: Packets can be sent to any destination in the NoC

Transpose-1: Packets are sent from (x, y) to (N-y+1, N-x+1)

Transpose-2(Bit reverse): Packets are sent from (x, y) to (y, x)

Shuffle: Packets are sent from (x, y) to ((x+N-1) mod N, (y+N-1) mod N))

Tornado: Packets are sent from (x, y) to ((x+([k/2]-1) mod N), (y+([k/2]-1) mod N))

Regional Uniform: A uniform traffic pattern that favours sending packets to nearby routers

Figures 5.1 illustrate how a regional uniform traffic pattern separates an 8x8 mesh NoC into 4 x 4 regions. This type of traffic has a higher likelihood as NoC organizations would likely to have cores that communicate with the closer NoC cores. Each region is organized as 4 x 4.

Figure 4 shows the organization of the regions. This type of traffic can be expanded to a 16 x 16 mesh. In our experiment, 80% of the traffic is sent within the region with the remaining 20% sent to anywhere in the NoC.

5.3 Performance Evaluation

One of the performance parameters used in NoCs is the average latency. By comparing average latencies between different traffic type and different NoC router architectures, an optimal architectural design can be determined. Average latency is the summation of the latency values of all the packets injected in the NoC divided by the number of packets injected.

Latency is simply calculated by

$$latency_{packet} = t_{arrival} - t_{injection} \qquad \text{Eq. 5.1}$$

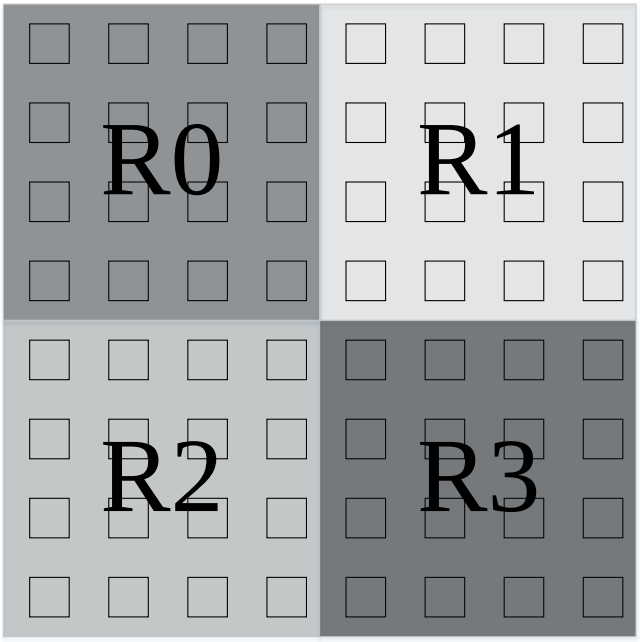


Figure 5-1 - Regional Uniform Traffic Regions

In our CAAR simulator, the source would embed the injection time into every flit of the packet. Once the packet's last flit reaches the sink core, its latency can be calculated and added to the running total.

Since average latency may not illustrate the worst case scenario for NoC performance, maximum latency can also be measured in the NoC and taken into account while predicted the NoC performance. To measure maximum latency, each sink core would track the highest latency packet received and send all the information to the simulator. The simulator would check all the sink values and determine the highest latency packet received.

5.4 Packet Tracking

Packet tracking is the process of keeping track of all the flits to ensure that all the flits and packets follow a path from the source to the destination without backtracking. This is important for the simulator to track while experimenting with any micro-architectural changes to the NoC router. For the simulator to track all the packets, a tracking process reads all the output flits at all the outputs of all the routers and sources. Every source keeps a list of all the flits and packets injected in the NoC. When a new packet is injected from a source core, each flit creates a new list and saves the source ID. Whenever a packet is being transmitted from a router, it compares its flit ID and adds the new router ID to the list. All lists are implemented as a linked list data structure which is dynamically expandable allowing millions of flits to be tracked by the simulator.

The process to track all the flits is illustrated below:

i. The source will read the current excitation time and embed the execution time value into the packet being injected into the NoC.

ii. The value will not change as the packet transverses all the NoC routers.

iii. Once the packet arrives at the sink, the sink waits until the tail flit arrives. Once the tail flit arrives, it will subtract the current execution time with the injection execution time to determine the latency.

5.5 Experimental Setup

NoC performance of the NoC router architecture is tested by the average latency of packets traversing the mesh NoC. The simulator is setup to allow experimental testing of many different configurations such as mesh topology size, injection rates, traffic patterns, length of the packets, the number of VCs, etc.

5.5.1 Mesh Topology Size

Our CAAR modeling experiment includes various size topologies. Since the industry is always looking to increase the number of cores and complexity of NoCs, smaller size NoCs such as 4 x 4 may not be very useful as today. We evaluated 8 x 8 and 16 x 16 mesh to determine the usefulness of our proposed CAAR based router and NoC.

5.5.2 NoC Simulation Setup

A transaction level SystemC simulator has been developed to model the microarchitecture of the router described in the previous section. The pipelined NoC router

consists of two cycles, one for VA/SA and the second for Crossbar switch transversal. An additional cycle is required for Link Transversal. Different routing algorithms have been used including DOR, Local Adaptive and destination based regional adaptive techniques. Fully adaptive algorithm is based on Duato's methodology, which employs an escape VC to break the deadlock [19]. In our experimental setup, the router uses 8 VCs with 5 flits per VC for the static sized VC buffer and various sized dynamically managed buffer. During evaluation, the simulator is initialized for 10,000 cycles and capable of capturing the results for 100,000 cycles afterwards. We evaluate each traffic pattern by employing a burst-based traffic injection process. We evaluate both average latency and maximum latency. As described earlier in section 5.3.3, maximum latency demonstrates the worst case scenario which is important to latency sensitive applications.

5.6 Evaluations

The Microsoft Excel software is used to produce the graphs for our evaluation results.

5.6.1 Small NoC (4x4)

We start to evaluate a small 4x4 mesh NoC. Since CAAR involves a prioritization technique, we do not expect CAAR to perform significantly different than RCA or any other regional congestion awareness. The average latency results are shown left along with the maximum latency on the right side.

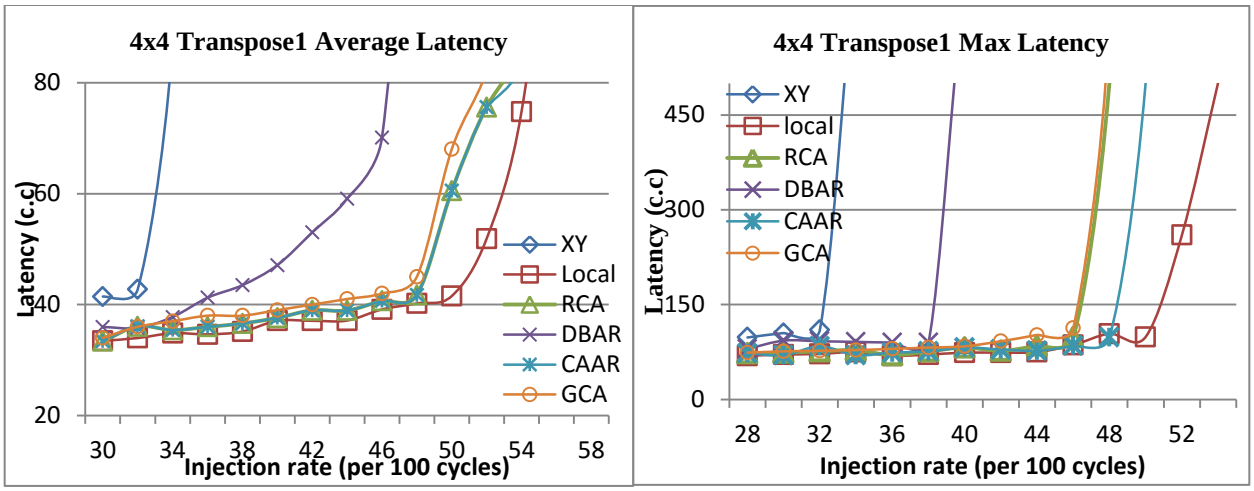


Figure 5-2 – 4x4 Transpose-1 Traffic

For Transpose-1 traffic and 4x4 mesh topology results of Figure 5-2, we can observe that CAAR has a slightly better maximum-latency over RCA. This is due to prioritization of packets sent between the corners of a mesh NoC. The results from Figures 5-2 and 5-4 show that Transpose-2 traffic has a slightly better average latency than RCA since there is less long distance packets being transmitted as compared to Transpose-1.

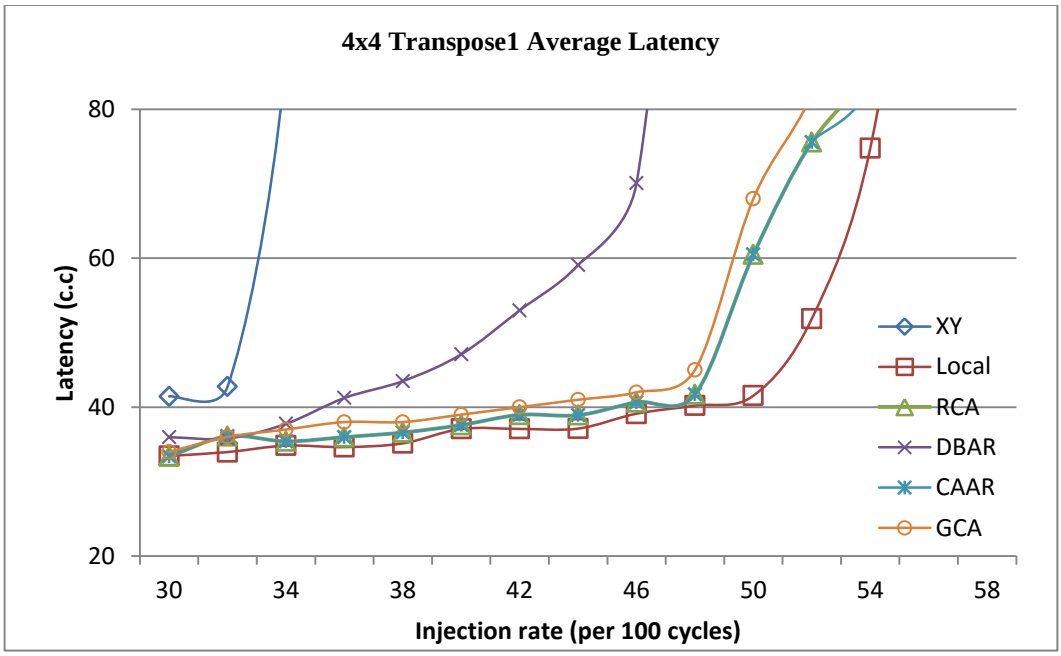


Figure 5-3 – Details of 4x4 Transpose-1 Average Traffic

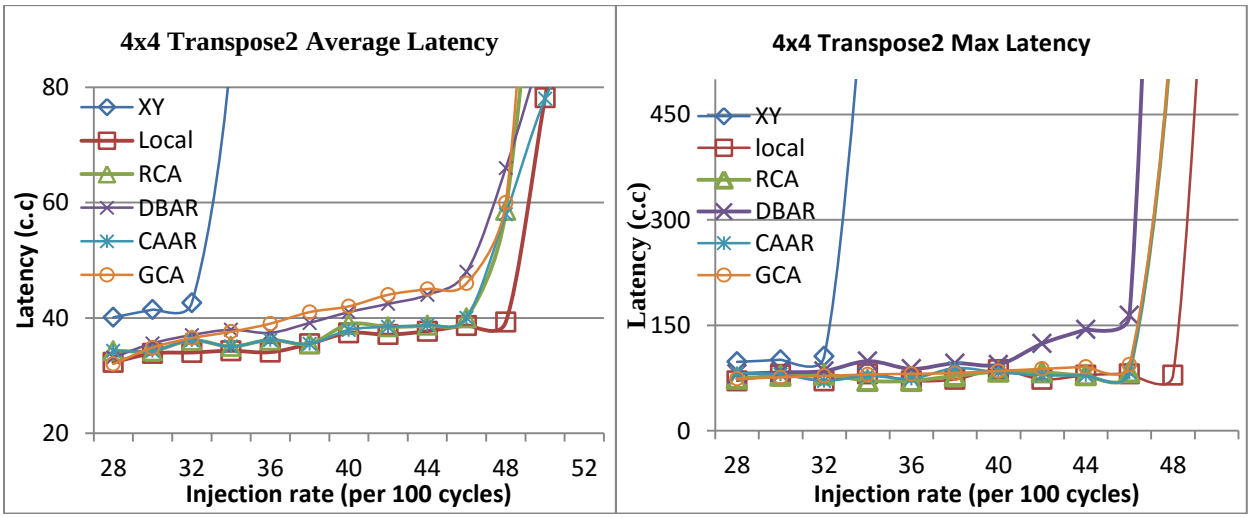


Figure 5-4 – 4x4 Transpose-2 Traffic

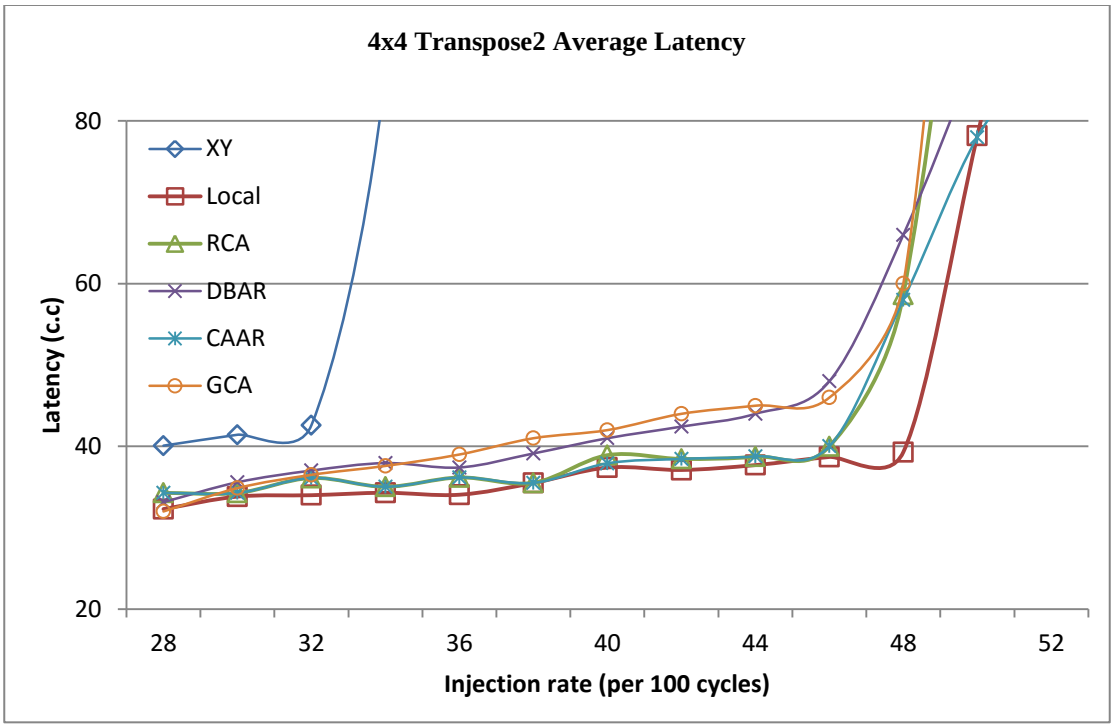


Figure 5-5-4x4 Transpose-2 Average Traffic

As expected, there would not be any improvement for shuffle traffic as all the destinations are within half the maximum distance of the mesh. Surprisingly XY routing performance is significantly better when enough number of VCs are available.

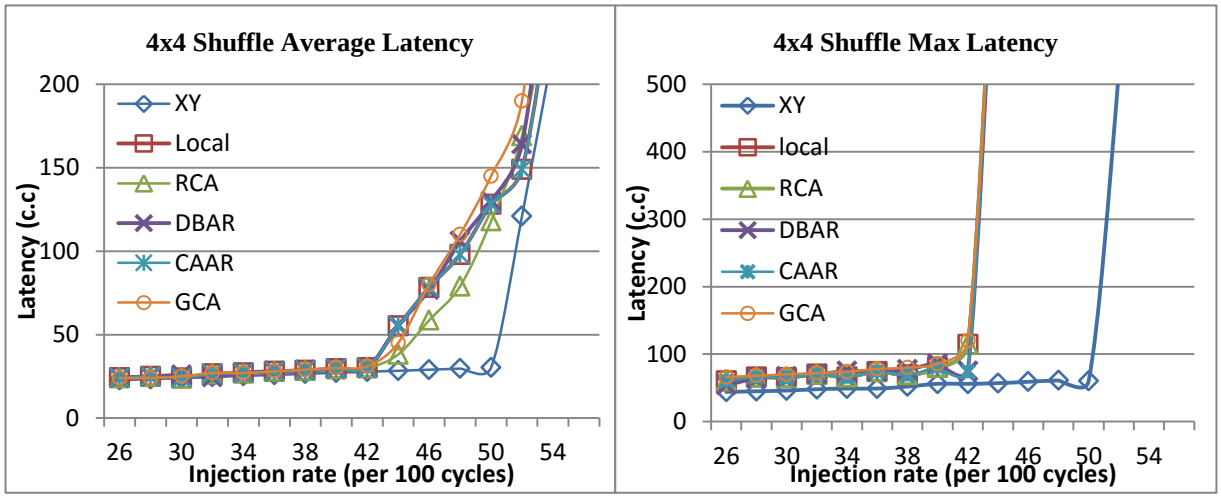


Figure 5-5 – 4x4 Mesh with Shuffle Traffic

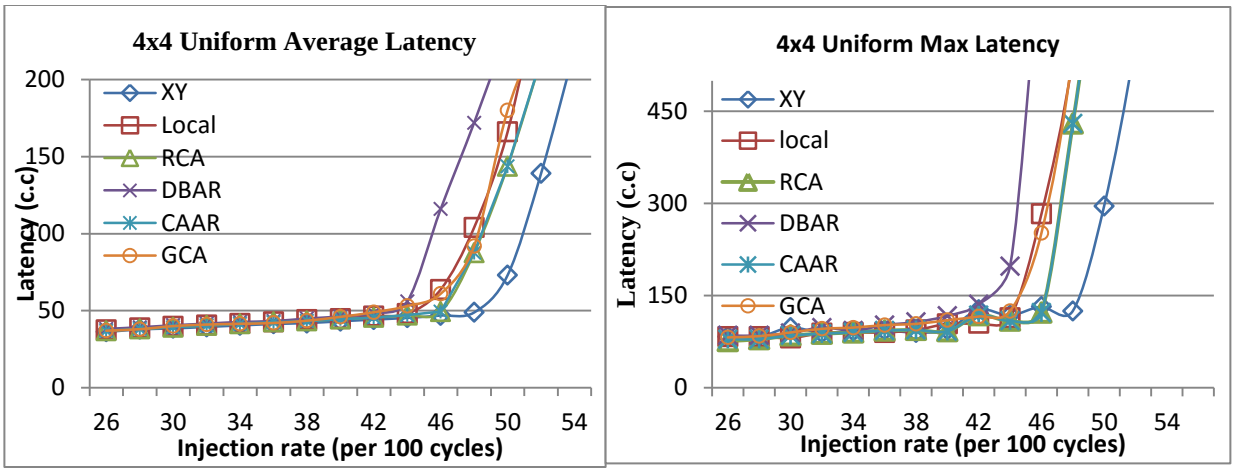


Figure 5-6 – 4x4 Uniform Traffic Latency

Under uniform traffic especially for a small 4x4 mesh, CAAR nor adaptive routing is expected to perform much better than XY routing. CAAR performed the same as RCA as there is a very low chance prioritization is required within uniform traffic. Uniform Regional traffic is not evaluated under the 4x4 mesh since it would have the same result as uniform traffic as the region size is 4x4.

5.6.2 Medium Size NoC (8x8)

In this section, we evaluated CAAR for an 8x8 mesh NoC with 8VCs. CAAR shows moderate improvement over RCA for transpose traffic as long distance packets have benefited from CAAR. Similar to 4x4, CAAR is able to improve throughput for transpose traffic. As shown in Figure 5-8, CAAR is able to gain small improvement with Transpose-1 traffic in contrast to 4x4 mesh NoC. Similarly CAAR has a larger performance gain with Transpose-2 traffic pattern (see Figure 5-9).

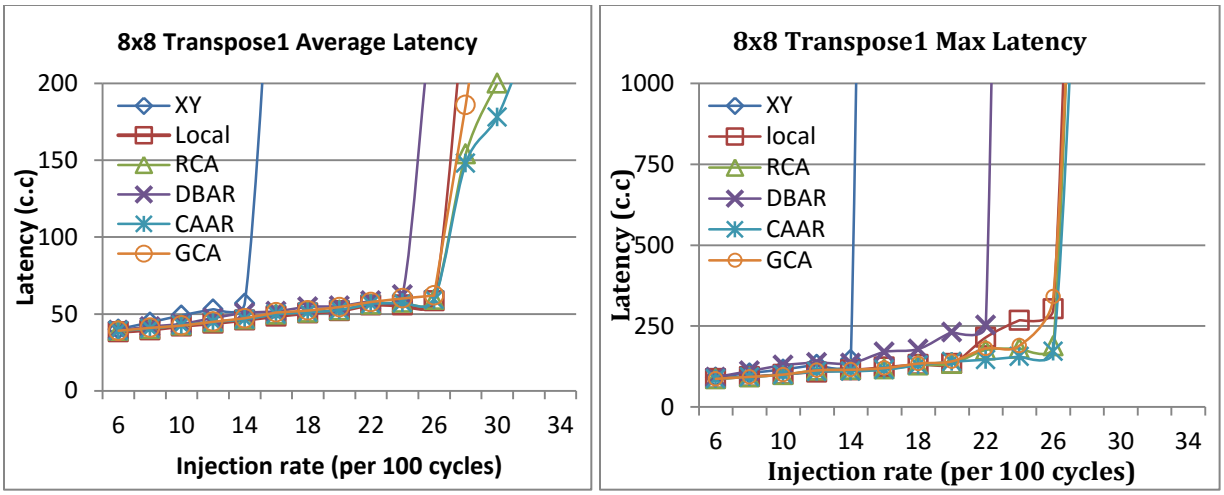


Figure 5-7 – 8x8 Transpose-1 Traffic

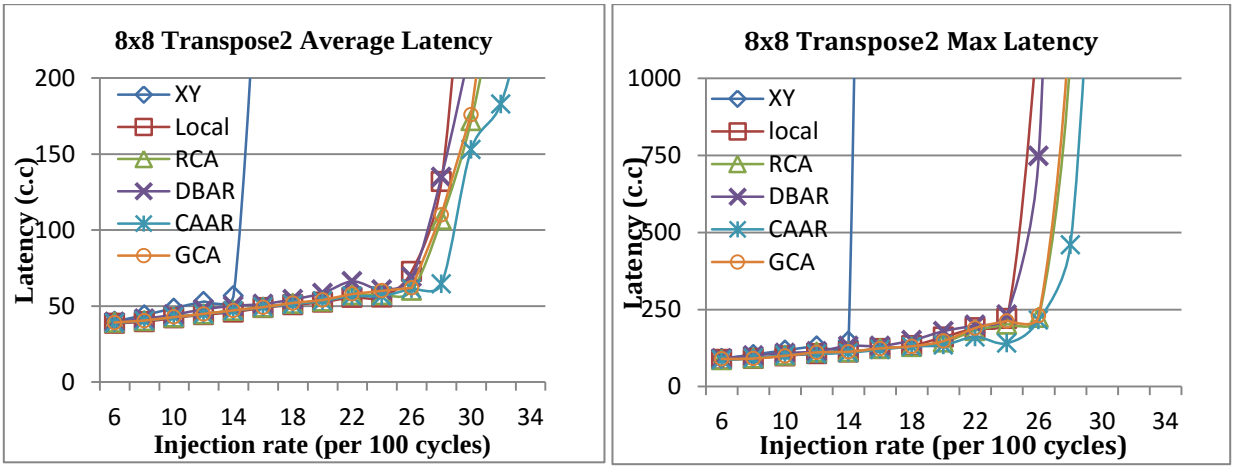


Figure 5-8 - 8x8 Transpose-2 Traffic

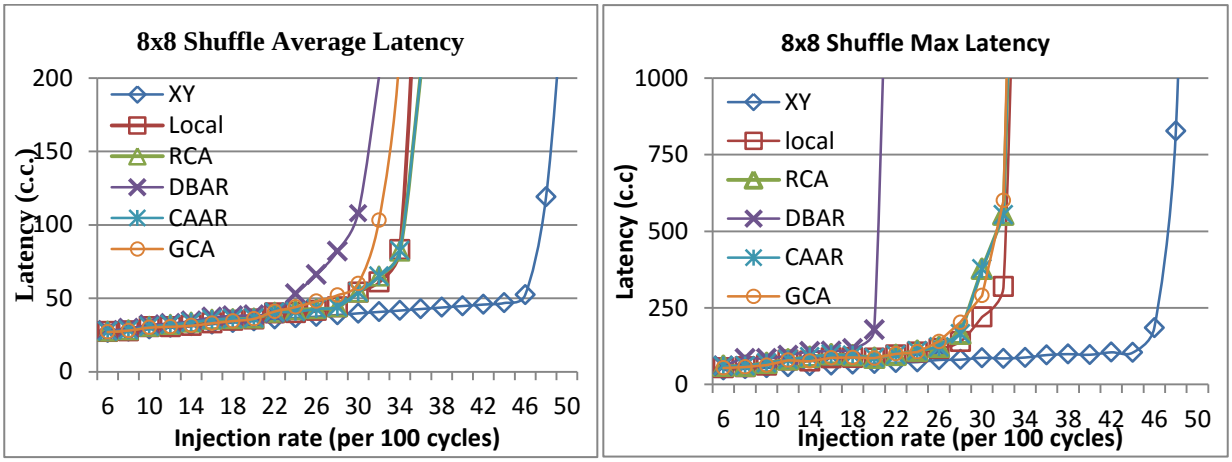


Figure 5-9 - 8x8 Shuffle Traffic

Similarly with shuffle traffic as shown in Figure 5-10, CAAR performed equally as good as RCA with XY performing significantly better. Under uniform traffic for an 8x8 mesh, average latency remains the same between RCA and CAAR but RCA performed slightly better than CAAR as shown in the results of Figure 5-11. This is due to an increased prioritization of long distance packets.

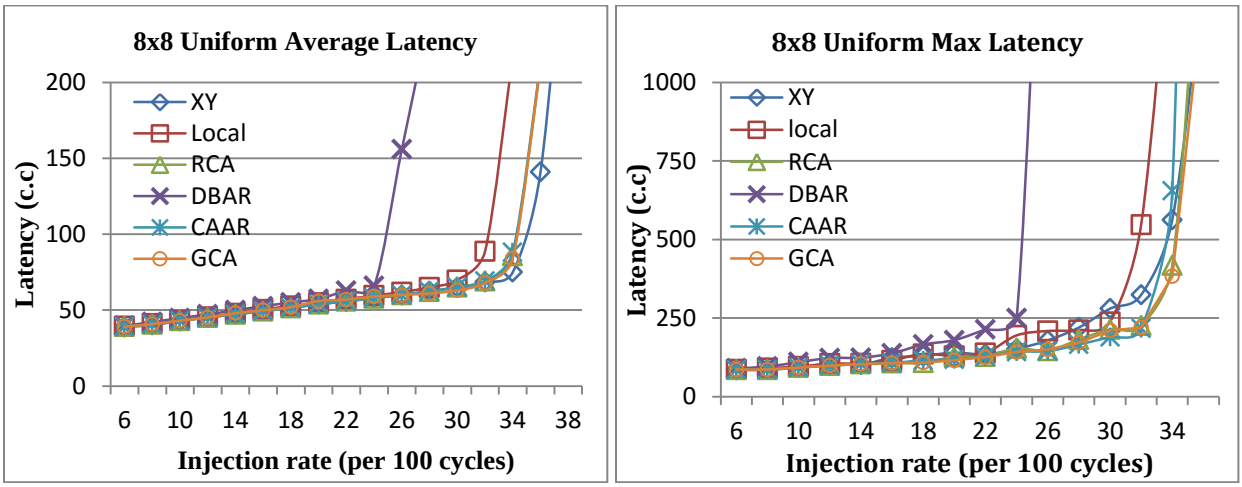


Figure 5-10 - 8x8 Uniform Traffic

Finally in figure 5-12, we present the evaluation results for a regional uniform traffic. Under this traffic condition, there is approximately 10% improvement in the case of CAAR when compared with RCA. With a small amount of packets that can be prioritized, the allocators are able to improve performance for the long distance packets.

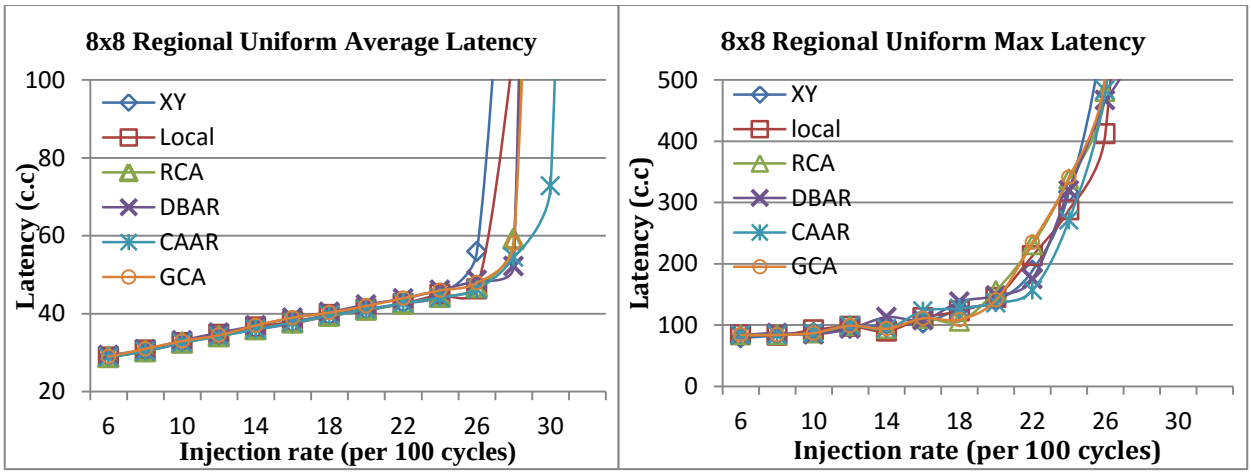


Figure 5-11 - 8x8 Regional Uniform Traffic

Figures 5-13 and 5-14 shows the results for regional uniform traffic on a larger scale to illustrate and highlight the better performance of CAAR based routing as compared to other schemes.

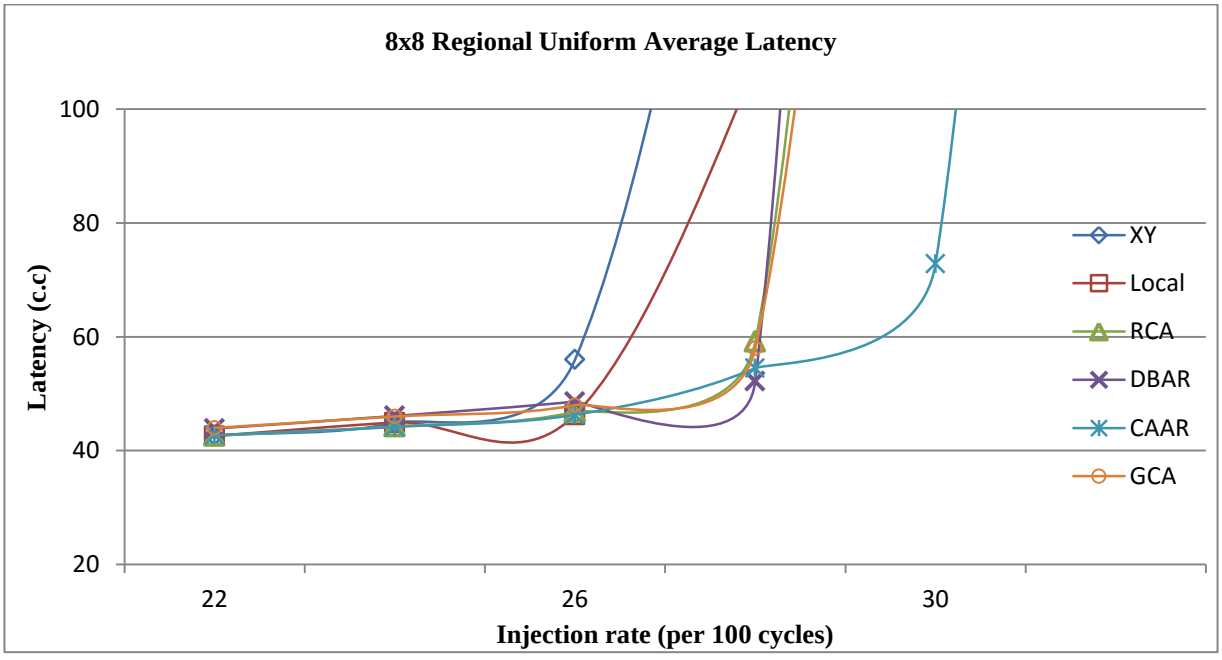


Figure 5-13 – Larger Scale of 8x8 Regional Uniform Traffic Average Traffic

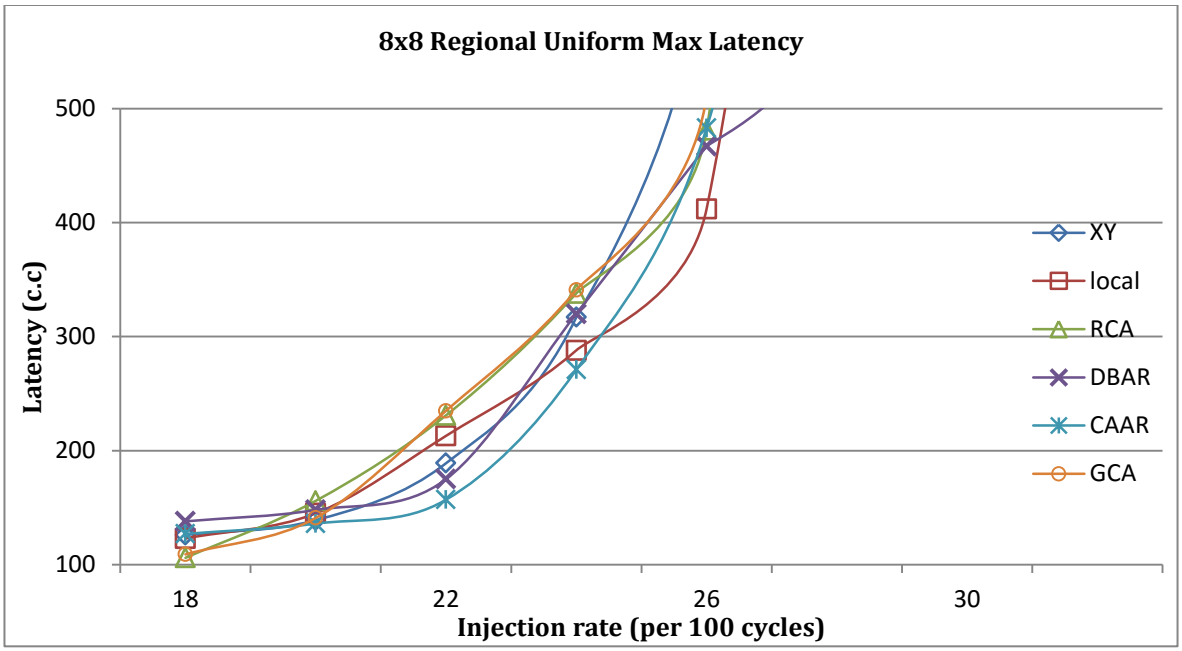


Figure 5-14 - Larger Scale of 8x8 Regional Uniform Traffic Max Traffic

5.6.3 Large Size NoC (16x16)

In this section the results are presented for 16x16 size mesh NoC in Figures 5-15 to 5-19. In a large size mesh NoC and for Transpose traffic, RCA shows a similar throughput as of CAAR but performs slightly better in the case of latencies when the NoC begins to saturate.

For the 16x16 mesh NoC and shuffle traffic, CAAR does show some improvements over RCA although the maximum for RCA is slightly better than CAAR the point of saturation.

Under uniform traffic, CAAR showed more improvement as compared to RCA. In terms of average latency, the deterministic XY routing performs better than RCA but CAAR performs better than both of them.

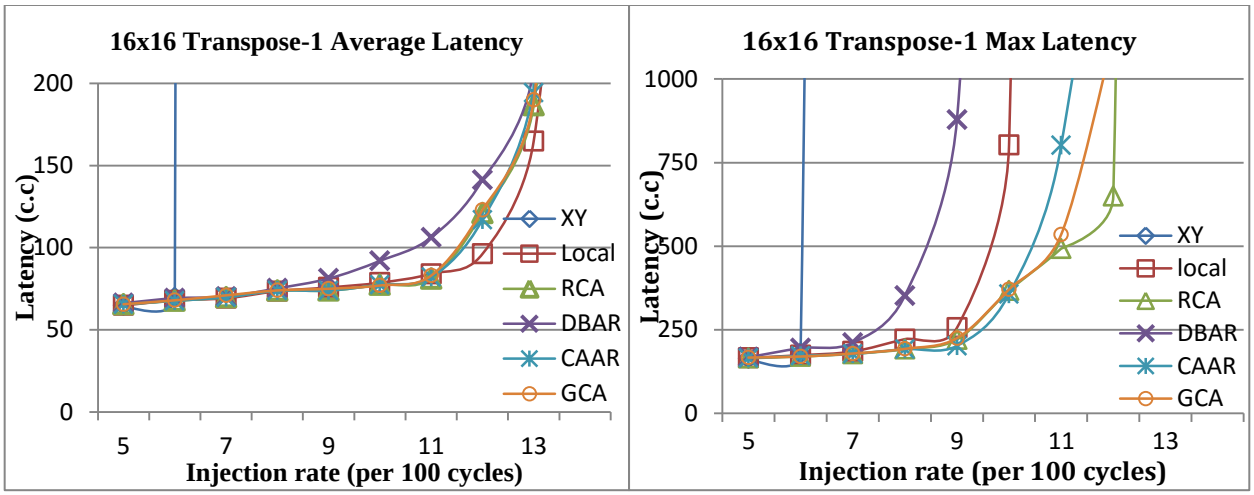


Figure 5-12 – 16x16 Transpose-1 Traffic

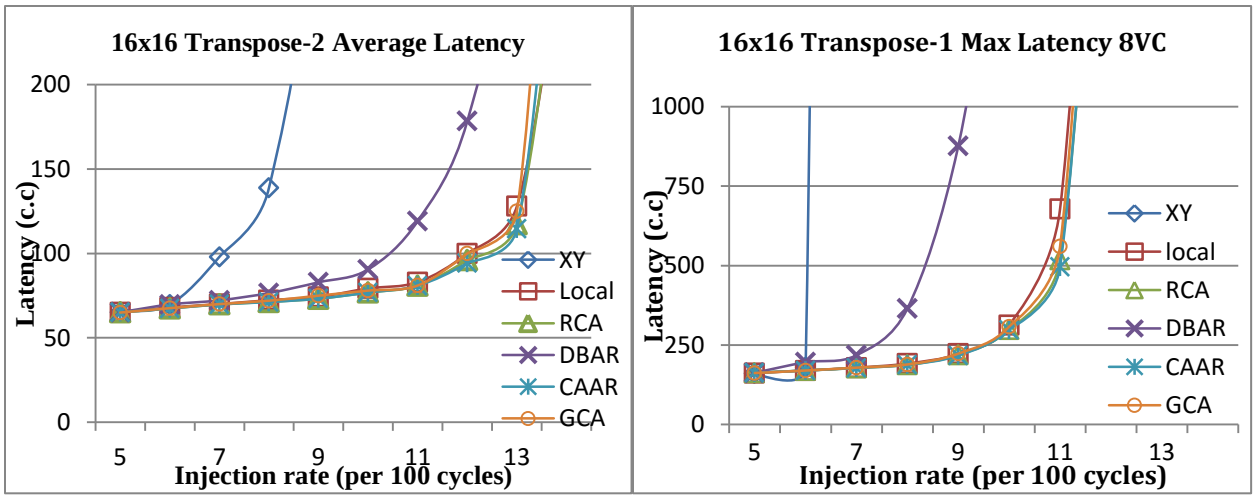


Figure 5-13- 16x16 Transpose-2 Traffic

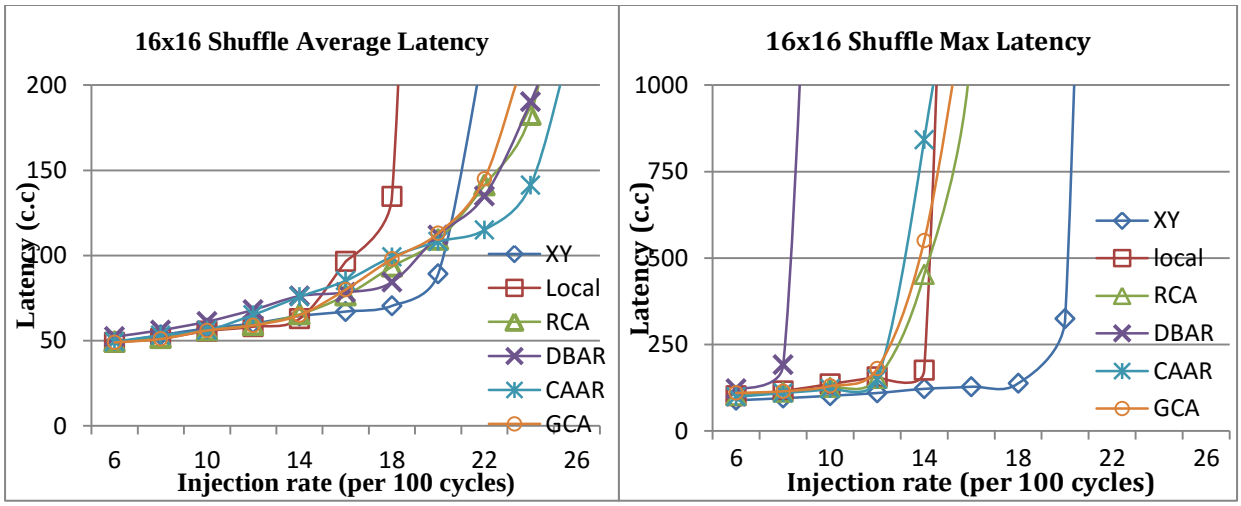


Figure 5-14 - 16x16 Shuffle Traffic

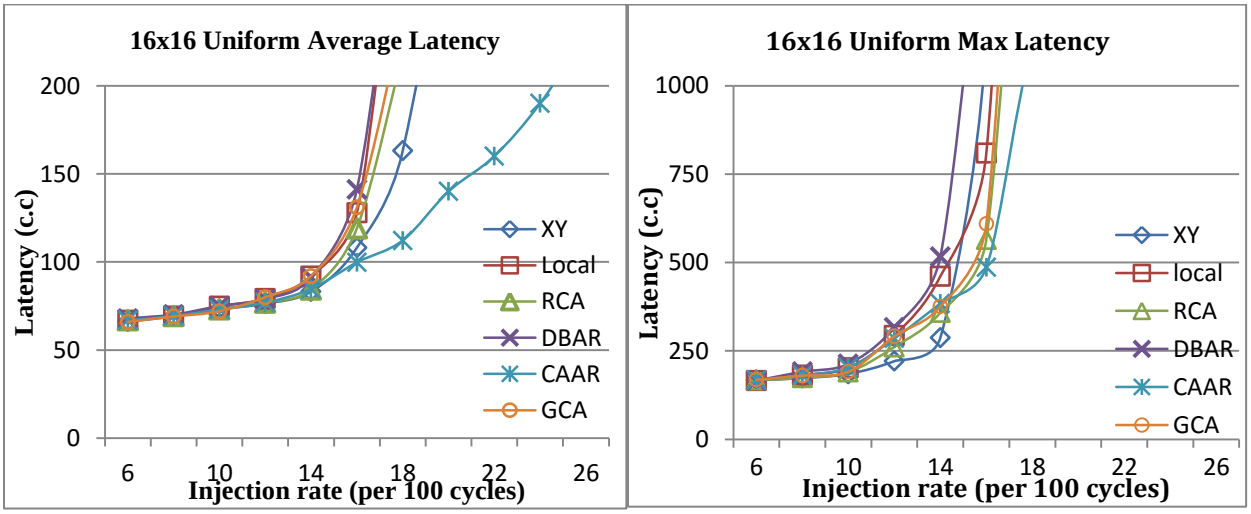


Figure 5-15 - 16x16 Uniform Traffic

Finally, in the case of regional uniform traffic, both RCA and CAAR performed similarly as most the traffic does not include any packets traveling from one corner of the NoC to an opposite corner.

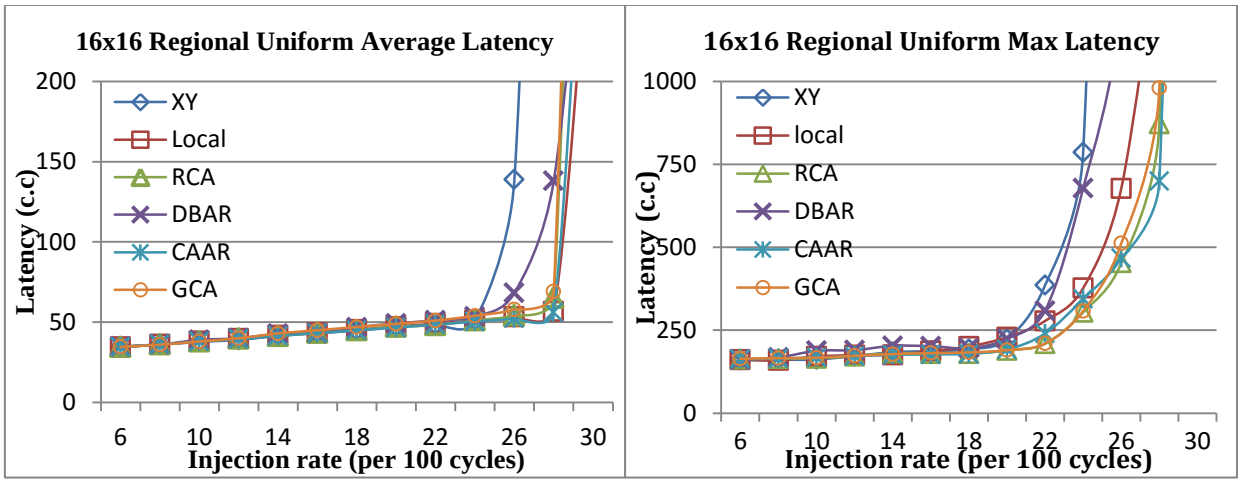


Figure 5-16 - 16x16 Regional Uniform Traffic

5.7 Hardware Modeling

In this section, we evaluate the power and area overhead associated with the additional hardware required for CAAR implementation. We'll make a comparison between the baseline adaptive routing with our CAAR implementation. The implementation of CAAR is done by using System Verilog.

5.7.1 Chip Area Estimation

Chip area estimation is the process of finding the area required for each router on the NoC chip. It is necessary to find out which component takes up more IC space and the changes in the chip size of the NoC router when more hardware is added to the router. For example, congestion awareness requires additional logical components that occupy more area space. Area estimation tools such as Synopsys Design Vision will be able to generate a report of the IC area usage.

5.7.2 Power Estimation

Hardware components require power to operate. Depending on the size and type of logic used in CAAR implementation, power usage can increase significantly. It is necessary to measure power usage of each router component, especially the power consumed by additional congestion awareness components.

5.7.3 Evaluation

We employed Synopsys Design Vision to analyze the area and power usage of our proposed CAAR router. We have used Nangate's Open Cell Library with a 15nm process at 1V [38]. The area and power usage estimates are presented in Table-1 for CAAR as compared to RCA and DBAR.

Table 5-1 – Hardware Area Usage

	Buffer	Other Logic	Allocator	Crossbar	Total
Baseline Local Adaptive	7598□□^{2}	682□□^{2}	3976□□^{2}	2735□□^{2}	14991□□^{2}
RCA	7598□□^{2}	1842□□^{2}	3976□□^{2}	2735□□^{2}	16151□□^{2}
DBAR	7598□□^{2}	2752□□^{2}	3976□□^{2}	2735□□^{2}	17061□□^{2}
CAAR	7598□□^{2}	1486□□^{2}	5171□□^{2}	2735□□^{2}	16990□□^{2}
GCA	7598□□^{2}	2202□□^{2}	3976□□^{2}	2735□□^{2}	16511□□^{2}

Table 5-2 – Power Usage

	Buffer	Other Logic	Allocator	Crossbar	Total
Baseline Local Adaptive	4.04mW	1.66mW	2.31mW	2.00mW	10.01mW
RCA	4.04mW	2.06mW	2.31mW	2.00mW	10.41mW
DBAR	4.04mW	3.02mW	2.31mW	2.00mW	11.37mW
CAAR	4.04mW	1.94mW	2.86mW	2.00mW	10.84mW
GCA	4.04mW	2.18mW	2.31mW	2.00mW	10.53mW

For an 8VC implementation, CAAR has an additional overhead of 5.2% over RCA and 13.3% over the baseline adaptive router. We believe this is justified as it is only a small overhead to improve the performance for far distance packets. Other proposed designs also have an overhead over RCA and the baseline adaptive router. CAAR has a power overhead of 4.1% over RCA and 8.3% over the baseline locally adaptive router. Compared to DBAR, CAAR has slight advantage in terms of both area usage and power consumption. DBAR would have higher power consumption every clock cycle as it needs to update all the destination information in the entire table. GCA has better hardware usage but is not as effective in terms of performance improvement over RCA.

Although there is a small increase of IC area and power consumption, congestion aware NoC routers are designed for high throughput applications where performance is the most important. For parallel high performance computing systems, a small increase in NoC router hardware is not significant as compared to the IC area size of the CPU cores.

5.8 Summary

This chapter provided the evaluation methods and results related to the performance of CAAR in relation with locally and regional adaptive routing. In this chapter, the methods of evaluation are identified along with the types of synthetic traffic used to test the proposed CAAR router. The results are illustrated along with a short reviewed to describe the situation. Furthermore, hardware overhead is also evaluated to determine CAAR's implementation in actual hardware. CAAR router allocator consumes much more chip area and there is need to investigate an area efficient CAAR allocator.

Chapter 6

Conclusions

This thesis presents a novel approach that improves NoC throughput by packet prioritization. The objective is to increase the NoC throughput by using congestion aware information. Congestion awareness information has already been applied for on-chip communication to improve NoC routing. We have expanded adaptive routing by employing regional congestion data to latency for packet routing. We have described the methodology of expanding the regional congestion awareness data to improve packet selection for both VC and switch allocators. A new methodology of Congestion Aware Adaptive Routing (CAAR) is designed to prioritize the packet/flit allocation that suffers the most latency while travelling between NoC cores. Moreover, to improve on hardware usage and to avoid any additional links between routers, CAAR removes the sideband network to transfer congestion data and instead adds the congestion information into the header flit of a packet.

Experiment and simulations have been conducted on various size mesh based NoCs ranging from 4x4 to 8x8 and up to 16x16. Our CAAR methodology experimental results demonstrate performance improvement for long distance packets, which are prioritized in congested NoC situations.

References

[1] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[2] L. Peh N. Enright Jerger, *On-Chip Network, Synthesis Lectures on Computer Architecture*.: Morgan & Claypool Publishers, 2009.

[3] G.-M. Chiu, "The Old-Even Turn Model for Adaptive Routing," *in IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729-738, July 2000.

[4] B. Grot, S.W. Keckler P. Gratz, "Regional Congestion Awareness for Load Balance in Network-on-Chip," *High performance Computer Architecture (HPCA)*, pp. 203-214, 2008.

[5] B. Lin R.S. Ramanujam, "Destination-Based Adaptive Routing on 2D Mesh Networks," *in Proceedings of Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, pp. 1-12, 25-26 October 2010.

[6] N. Enright Jerger, Z. Wang S. Ma, "DBAR: An Efficient Routing Algorithm to Support Multiple Concurrent Applications in Network-on-Chip," *in Proceedings of Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 413-424, 4-8 June 2011.

[7] T. Takabatake, "Simulations of NoC topologies for generalized hierarchical completely-connected networks," *in Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, vol. , no. , pp. 1-5, 20-22 Jun. 2011.

[8] A., Saoud, S. Achballah, "A Survey of Network-On-Chip Tools," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 4, no. 9, pp. 61-67, Dec. 2013.

[9] T., Mahadevan, S. Bjerregaard, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys*, vol. 38, no. 1, pp. 1-51, Jun 2006.

[10] P. Bogdan, G. Wei, C.Y. Tsui, R. Marculescu Z. Qian, "A Traffic-aware Adaptive Routing Algorithm on a Highly Reconfigurable Network-on-Chip Architecture," *in Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, vol. , no. , pp. 161-170, October 2012.

[11] C., Bagherzadeh, N. Wang, "Design and evaluation of a high throughput QoS-aware and congestion-aware router architecture for Network-on-Chip," *in proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pp. 457-464, 15-17 Feb. 2012.

[12] S. Liu, A. Jantsch, and Z. Lu, "Analysis and Evaluation of Circuit Switched NoC and Packet Switched NoC," *in the Proceeding of the 16th Euromicro Conference on Digital System Design*, pp. 21-28, 4-6 Sept. 2013.

[13] P. Gratz, A. Sprintson M. Ramakrishna, "GCA: Global Congestion Awareness for Load Balance in Network-on-Chip," *in Proceeding of the 7th IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pp. 1-8, 21-24 April 2013.

[14] C.A. Nicopoulos et al., "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-

Chip Routers," *in Proceeding of International Symposium on Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM*, pp. 333-346, 9-13 Dec. 2006.

[15] W. J. Dally and H. Aoki, "Deadlock-free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466-475, April 1993.

[16] A. West, S. Moore R. Mullins, "Low-Latency Virtual-Channel Routers for On-Chip Networks," *in Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA). Munich, Germany.*, pp. 188-197, 19-23 June 2004.

[17] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions of Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055–1067, October 1995.

[18] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 8, pp. 841–854, August 1996.

[19] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, December 1993.

[20] T. Lehtonen, J. Plosila V. Rantala, "Network on Chip Routing Algorithms," *Turku Centre for Computer Science*, August 2006.

[21] K. Jain, S.K. Singh, A. Majumder, and A.J Mondai, "Problems encountered in various arbitration techniques used in NOC router: A survey," *in Proceeding of the International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV)* , pp. 62-67, 29-30 Jan. 2015.

[22] W. Dally and D. Becker, "Allocator implementations for network-on-chip routers," *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, November 2009.

[23] N McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *in IEEE Transactions on Networks*, vol. 7, no. 2, pp. 188-201, Apr. 1999.

[24] Q.-A. Zeng, W.-B. Jone and M. Li, "DyXY – A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for network on Chip," *in proceeding of the 43rd Design Automation Conference. San Francisco, CA*, pp. 849-852, July 2006.

[25] J., Marculescu and R. Hu, "DyAD-Smart Routing for Network-on-Chip," *in Proceeding of 41th Design Automation Conference. San Diego, CA.*, pp. 260-263, June 2004.

[26] D. Park, T. Theocharides, N. Vijaykrishnan, C.R. and Das J. Kim, "A Low Latency Router Supporting Adapitivity for On-Chip Interconnects," *in Proceeding of 42th Design Automation Conference*, pp. 559-564, June 2005.

[27] V. Catania, M. Palesi, D. Patti and G. Ascia, "Implementation and Analysis of a New Selection Strategy for Adaptive Routing in Networks-on-Chip," *IEEE Transaction in Computers*, vol. 57, no. 6, pp. 809-820, Apr. 2008.

[28] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Q-learning based congestion-aware routing algorithm for on-chip network," *in Proceeding of the 2nd International Conference on Networked Embedded Systems for Enterprise Application*, pp. 1-7, 8-9 Dec. 2011.

[29] L.Clermidy, F. Moraes, and F. Tedesco, "A Monitoring and Adaptive Routing Mechanism for Qos Traffic on Mesh NoC Architectures," *in Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). Grenoble, France,* pp. 109-117, June 2009.

[30] M., Fu, W., Chen, T., Hu, W., Wu, and M. Yuan, "CABSR: Congestion Agent Based Source Routing for Network-on-Chip," *in Proceeding of the Intl Symp on Cyberspace Safety and Security on High Performance Computing and Communications*, pp. 669-676, 20-22 Aug. 2014.

[31] H.-L. Chen, Y.R., Tung, S.-Y., Hsiung, P.-A., Chen, and S.-J. Chao, "Congestion-aware scheduling for NoC-based reconfigurable systems," *in Proceeding of Design, Automation & Test in Europe Conference & Exhibition (DATE) on* , vol. pp. 1561-1566, 12-16 Mar. 2012.

[32] M. Daneshtalab, M. Ebrahimi, J. Plosila, and H. Tenhunen, "CARS: Congestion-aware request scheduler for network interfaces in NoC-based manycore systems," *in Proceeding of Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 1048-1051, 18-22 Mar. 2013.

[33] D. Zydek and H. Selvaraj, "Processor Allocation Problem for NoC-Based Chip Multiprocessors," *in the Proceeding of the Sixth International Conference on Information*

Technology: New Generations (ITNG), pp. 96-101, 27-29 Apr. 2009.

[34] R. Das, O. Mutlu, T. Moscibroda, and C.R. Das, "Application-aware prioritization mechanisms for on-chip networks," *in the Proceeding of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 280-291, 12-16 Dec. 2009.

[35] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414-421, Sept.-Oct. 2005.

[36] J. Diemer, R. Ernst, and M Kauschke, "Efficient throughput-guarantees for latency-sensitive networks-on-chip," *in Proceeding of 15th Asia and South Pacific on Design Automation Conference (ASP-DAC)*, pp. 529-534, 18-21 Jan. 2010.

[37] Accellera. (2015) SystemC. [Online]. http://accellera.org/downloads/standards/systemc

[38] J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, J. Michelsen and M. Martins, "Open Cell Library in 15nm FreePDK Technology," *in Proceedings of the International Symposium on Physical Design (ISPD)*, pp. 171-178, 29 Mar. 2015.