Adaptive and Distributed Networks-on-Chip for Mixed Criticality Systems

Zur Erlangung des akademischen Grades einer

DOKTORIN DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

DISSERTATION

von

M.Sc. Nidhi Anantharajaiah

Tag der mündlichen Prüfung:	24.04.2024
Hauptreferent:	Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent:	Prof. Dr. sc.techn. Andreas Herkersdorf

In Loving Memory of my Grandmother Smt.Vilasini Sanathkumar (Eramma)

Zusammenfassung

Durch das Ende des Mooreschen Gesetzes besteht die Notwendigkeit Transistoren effizienter zu nutzen um die Kosten zu senken. Mit dem jüngst erfolgten Wegfall von Dennard's Scaling ist es zusätzlich notwendig geworden mehrere effizientere Prozessor-Kerne anstelle eines einzelnen Kerns zu verwenden. Mit dem Wachstum der Anzahl vorhandener Prozessorkerne wurden die traditionell genutzten Kommunikationsinfrastrukturen basierend auf Bussen zum limitierenden Faktor für die insgesamte Leistungsfähigkeit. Network-on-Chips wurden hierzu als skalierbare Kommunikationsinfrastruktur vorgeschlagen, die modular und flexibel ist. Gleichzeitig stellen Echtzeitanwendungen die Mehrheit der im Bereich der eingebetteten Systeme zum Einsatz kommender Anwendungen dar. In solchen Systemen sind strikte Fristen üblich wodurch zusätzliche Anforderungen an die On-Chip-Kommunikation gestellt werden. Zu den Herausforderungen, mit denen mixed criticality NoCs konfrontiert sind, gehören die Unterstützung von echtzeitkritischen Anwendungen, die Ermöglichung einer effizienten gemeinsamen Nutzung von Ressourcen und die Unterstützung von mixed criticality Datenverkehr bei gleichzeitiger Erfüllung der nicht-funktionaler Anforderungen. Echtzeit-MPSoCs sind unterschiedlichen anwendungsspezifischen Anforderungen und unterschiedlichen Arbeitsprofilen zur Laufzeit ausgesetzt, was den Bedarf der Entwicklung adaptiver NoCs motiviert. Um diese Herausforderungen zu bewältigen, wird in dieser Arbeit die inhärente verteilte Natur eines NoCs genutzt um dessen Anpassungsfähigkeit und Flexibilität zu erhöhen.

Diese Arbeit präsentiert ein neuartiges DREAM NoC: A **D**istributed **R**einforcement learning **E**nabled **A**daptive **M**ixed-Critical Network-on-Chip und ein unterstützendes DREAM-Framework. Das vorgeschlagene Routing-Schema ist topologieunabhängig und findet bessere Routen zur Laufzeit der Anwendung auf der

Grundlage von Schwankungen des Datenverkehrs. Die Routen werden auf der Grundlage einer Kombination aus kurz- und langfristigen Informationen über den Netzwerkzustand berechnet, wobei interne Routing-Tabellen im Laufe der Zeit auf verteilte Weise im NoC aktualisiert werden. Diese Arbeit stellt vor wie das DREAM NoC verwendet werden kann um den Datenverkehr innerhalb von Partitionen zu isolieren, und wie das DREAM-Framework zur Unterstützung einer flexiblen räumlichen Isolierung eingesetzt wird. Das Framework sieht eine Lernzeit vor, um Routen in Regionen zu entdecken, deren Topologien zur Entwurfszeit unbekannt sein können. Zusätzlich wird ein NoCs mit mehreren Ebenen zusammen mit unterstützenden adaptiven, überlastungsbewussten Routing-Mechanismen, die auf mixed criticality Datenverkehr abzielen vorgeschlagen und präsentiert.

Abstract

Due to the ending of Moore's Law there is a need to efficiently use transistors to reduce cost overhead. The recent ending of Dennard scaling has additionally motivated the need to use multiple more efficient cores instead of a single core. With the increase in the number of cores, traditional bus-based communication was becoming a limiting factor for performance. Network-on-Chips was proposed as a scalable communication infrastructure solution that is modular and flexible.

Real-time applications occupy a majority in the embedded systems field and additionally impose constraints on such on-chip communication to meet certain stringent deadlines. Challenges which mixed criticality NoCs face include supporting real-time critical applications, enabling efficient resource sharing, and supporting mixed critical traffic while meeting non-functional requirements. Real-time MPSoCs are subjected to varying application specific demands and varying workloads at runtime, motivating the need to design adaptive NoCs. To address these challenges, in this work, the aim is to utilize the inherent distributed nature of a NoC, and propose increasing its adaptivity and flexibility.

This work presents a novel DREAM NoC: A **D**istributed **R**einforcement learning **E**nabled **A**daptive **M**ixed-Critical Network-on-Chip and supporting DREAM framework. The proposed routing scheme is topology agnostic and discovers better routes during application runtime based on traffic fluctuations. Routes are computed based on a combination of short and long-term information of the network state, and routing tables are updated in a distributed manner over time across the NoC. This work presents how the DREAM NoC can be used to isolate traffic within partitions, and using the DREAM framework to aid in flexible spatial isolation. The framework allocates a learning time period to discover

routes in regions whose topologies can be unknown at design time. Additionally, a multi-layered NoC and supporting adaptive congestion aware routing mechanisms targeting mixed criticality traffic is also proposed.

Preface

This thesis is about connections and communication. To be more specific, it is about the interconnection and communication between components that form complex System-on-Chips. It investigates how adaptability and flexibility can be integrated into such a communication infrastructure to improve performance while ensuring application requirements are met. A tiny sliver in the vast area of on-chip communication is discussed here. And I hope that through this book, dear reader, I can communicate and connect with you.

The work presented here was conducted when I was a researcher at the Institut für Technik der Informationsverarbeitung (ITIV) at the Karlsruhe Institute of Technology (KIT). I am very grateful for the time I spent here and the connections I made. I want to first thank my supervisor, Prof. Jürgen Becker, for accepting me as his student and for giving me this opportunity. I am very grateful for his kindness, consistent support, trust, patience, freedom, flexibility, guidance, and motivation at every step of this journey. I want to especially thank him for being always supportive and encouraging of my little drawings.

I want to thank Prof. Andreas Herkersdorf for being my co-supervisor, for his support, guidance, and discussions throughout this time period. I want to thank the exam committee members, Prof. Ivan Peric, Prof. Ahmet Cagri Ulusoy, and Prof. Thomas Leibfried, for their support, feedback, and discussions. I want to especially thank Prof. Gert F. Trommer, for his time, feedback, and kind, encouraging words before and on the day of my exam.

I want to thank all the students I had the opportunity to supervise, for their support, numerous discussions, and long debugging sessions, which gave me that extra motivation to work hard on tough days. I am immensely grateful to have

had wonderful colleagues. Thank you very much for the countless discussions, collaborations, work trips, restaurant trips, movies, and DnD games, for being recipients of my silly drawings, for discussions after tough, chaotic days, and for keeping me sane.

I want to thank my parents, without whom none of this would have been even remotely possible. I want to thank them for always believing in me, for their unwavering patience, courage, support, and encouragement, and for sending their only child on her crazy adventures across the globe. I can never thank them enough. Finally, I want to thank my partner, who has always been a source of immense joy, support, patience, and encouragement. It made any task seem much easier to manage and allowed the thought that this was not impossible to do after all.

Nidhi Anantharajaiah
April, 2025

Contents

Acronyms

ACO	Ant Colony Optimization
AMBA	Advanced Microcontroller Bus Architecture
AP	Application Phase
ASIC	Application-Specific Integrated Circuit
BE	Best-Effort
CPU	Central Processing Unit
CS	Circuit Switching
DLP	Data-Level Parallelism
DNN	Deep Learning Neural Network
DoR	Dimension Order Routing
DREAM	Distributed Reinforcement learning Enabled Adaptive Mixed-critical
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous and Locally Synchronous
GPU	Graphics Processing Unit
GS	Guaranteed Service
GT	Guaranteed Throughput

HDL	Hardware Description Language
ILP	Instruction Level Parallelism
iNoC	Invasive Network-on-Chip
IoT	Internet-of-Things
ITRS	International Technology Roadmap for Semiconductors
KIT	Karlsruhe Institute of Technology
LBDR	Logic Based Distributed Routing
LP	Learning Phase
LUT	Look-Up-Table
MIMD	Multiple-Instruction, Multiple-Data
MPI	Message Passing Interface
MPSoC	Multi-Processor System-on-Chip
MCM	Multi-Chip-Module
MCS	Mixed Criticality Systems
NA	Network Adapter
NI	Network Interface
NoC	Network-on-Chip
NoP	Network-on-Package
OS	Operating System
OSI	Open Systems Interconnection
PCB	Printed Circuit Board
PE	Processing Element

PL	Programmable Logic
PS	Packet Switching
QoS	Quality-of-Service
RF	Radio Frequency
RL	Reinforcement Learning
RLP	Request-Level Parallelism
RR	Round-Robin
SAF	Store And Forward
SL	Service Level
SoC	System-on-Chip
SPMD	Single-Program Multiple-Data
TDM	Time Division Multiplexing
TLP	Thread-Level Parallelism
TLM	Tile Local Memory
TS	Time Slot
TSV	Through-Silicon Via
UD	Up*/Down*
VC	Virtual Channel
VCT	Virtual Cut Through
WC	Worst Case
WCET	Worst-Case Execution Time
WiNoC	Wireless Network-on-Chip

1 Introduction

"*Survival of the Fittest*" is a phrase made famous by naturalist Charles Darwin in his book "On the Origin of Species"[1]. Here, the term *fittest*, does not indicate strongest but it implies organisms that are better *adapted* to their local environment. Organisms that are the most adaptable to their local environment are more successful in surviving.

Advancements in technology are not dissimilar. Technological trends rise and fall, and it is prudent to be aware, investigate new techniques, and adapt accordingly. Recently, the trend of transistor count on a chip doubling every 18 to 24 months, more popularly known as Moore's law, ended, as illustrated in Figure. 1.1 [2]. Another observation is the ending of Dennard scaling, leading to the advent of multiple more energy efficient processors on a chip and limitations of parallel processing described by Amdahl's Law. Dennard scaling is a trend where the power density is constant for a given area of silicon even with the increase in transistor count due to the smaller dimensions of each transistor. The speed of the transistors increased while consuming less power. This trend ended around 2004. This impacted the microprocessor industry, leading to one processor being replaced with multiple more efficient processors. Resulting in an industry roadmap based on multi-processor-on-a-chip rather than a faster uni-processor. An industry example is Intel, in 2004 cancelled projects based on high-performance uni-processors [2].These trends have had an impact on the processor performance and have caused it to slow down. The improvement in processor performance is now doubling every 20 years instead of every 1.5 years, as seen between 1986 and 2003 and illustrated in Figure. 1.2 [2].

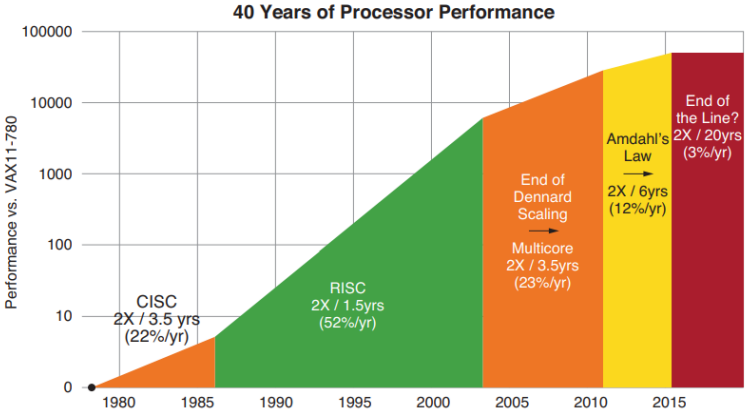


Figure 1.1: End of Moore's Law [2][3]

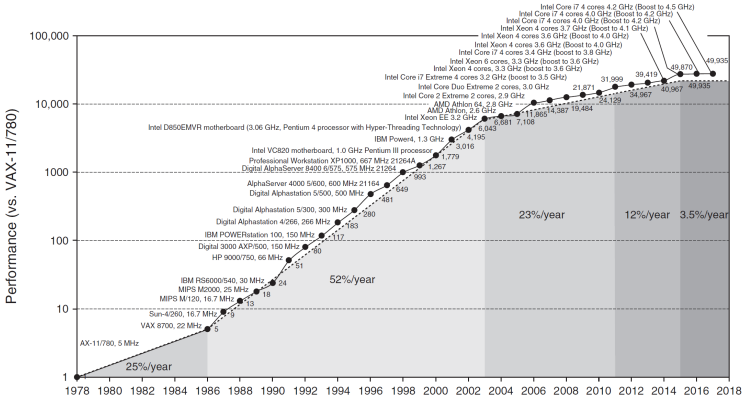


Figure 1.2: Growth in processor performance over 40 years [2]

The ever increasing demand for computational services, in parallel with the decrease in transistor scaling, has motivated the development of complex systems with improved performance. The increase in integration density in digital circuits has resulted in growing complex systems on a single chip. Multi-Processor Systems-on-Chip (MPSoCs) have evolved to contain 100s of processing cores. The number of cores in a single processor chip is continuously increasing. A latest example is AmpereOne [4] from Ampere, a 64 bit multi-core processor comprising of upto 192 cores. Future MPSoCs are expected to contain thousands of processing cores [5] as estimated by the International Technology Roadmap for Semiconductors (ITRS).

In recent years, System-on-Chip (SoC) designs comprising of "chiplets" have been gaining popularity [6]. Such SoCs comprise of multiple smaller, higher-yield and cost-effective chiplets which are assembled using advanced packaging technologies. Package level integration using multi-chip-module (MCM) is a promising approach to building large-scale systems by combining multiple smaller building blocks known as chiplets into a larger system. Instead of a conventional monolithic SoC, such architectures contain chiplets that are smaller, cheaper to develop and easier to reuse across multiple products. Chiplet based technology is being investigated in industry and academia[7].

Focusing on the communication on such systems, more than a decade ago, researchers identified that traditional bus-based communication on SoC was becoming a limiting factor for performance [8]. The global communication pattern was moving towards being fully distributed with minimal global coordination [8]. Researchers postulated that utilizing the layered design of reconfigurable micronetworks can achieve better performance on SoCs leading to the development of Networks-on-Chips(NoC) [8]. NoCs have been proposed as a scalable solution for the increasingly complicated communication requirements of MPSoCs.

NoCs are a modular, scalable and flexible communication infrastructure. They are increasingly popular in heterogeneous MPSoC architectures, interconnecting processing elements, DSPs, IP cores and memory units for example. A simple illustration is shown in Figure 1.3, where 16 heterogeneous tiles are connected

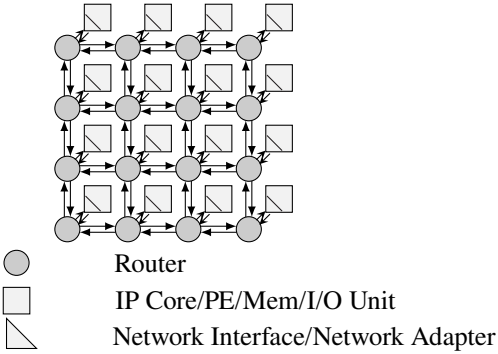


Figure 1.3: A 4x4 Mesh topology NoC

using a 4x4 mesh topology. Such platforms are commonly used in real-time multimedia processing applications. Real-time applications (RTAs) form a majority of applications in the embedded systems field [9]. This imposes constraints on the systems as real-time computation and communication need to meet certain stringent deadlines[9]. An important trend when designing real-time embedded systems is integrating components of different criticality on the same computing platform [10] to reduce overall cost. Criticality is *"a designation of the level of assurance needed against failure for a system component"*, as described by authors in [10] . And when a system comprises of components adhering to multiple distinct criticality levels, they are referred to as Mixed Criticality Systems (MCS).Many systems in automotive and avionic domains are moving towards MCS to meet stringent non-functional requirements[10]. These requirements can include overall cost, space occupied, weight, heat generation, and power consumption.

With the increase in the number of cores in computing platforms and the increase in applications sharing such a heterogeneous platform, the on-chip communication is becoming a critically shared resource. With the usage of NoCs in industry, a wide range of new challenges are introduced due to application demands along with cost constraints. Challenges that NoCs face include supporting real-time

critical applications, enabling efficient resource sharing and supporting mixed-critical traffic while meeting non-functional requirements. Performance of NoCs is especially vital for time-critical applications where Worst-Case Execution Time (WCET) requirements have to be met. Over the years, numerous techniques, methodologies and architectures across different computational layers of the on-chip interconnect have been proposed to address the numerous challenges.

Real-time MPSoCs are subjected to varying application specific demands and varying workloads at runtime, motivating the need to design adaptive NoCs. In mixed criticality systems where multiple applications that have varying requirements and criticality are sharing an interconnect, there is potential to make the NoC more adaptable to improve overall performance. Mixed critical applications can gain from runtime adaptivity.An ongoing research area is developing runtime adaptive NoCs while maintaining hard real-time guarantees [9].

The thesis takes a look at the inherent distributed nature of a NoC and propose increasing its adaptivity and flexibility at different levels. At the routing level, a reinforcement learning based algorithm which is topology agnostic is developed where optimal paths are discovered over time, resulting in a self-learning network. How such an algorithm can be used to aid in spatial isolation for MCS is illustrated by proposing a framework. The thesis refers to this contribution as DREAM NoC: A **D**istributed **R**einforcement learning **E**nabled **A**daptive **M**ixed-Critical Network-on-Chip. A supporting DREAM framework is proposed that uses such a self-learning NoC to aid in spatial isolation in MCS. The thesis also takes a look at improving the adaptivity and increasing scalability of multicast communication protocols by proposing a Block-Based multicast technique. Adaptivity and flexibility at the topology level are also investigated by proposing a multilayered NoC for MCS. A supporting congestion aware adaptive routing algorithms with criticality aware traffic distribution is proposed.The proposed features, techniques and framework focus on introducing adaptivity in NoCs in MCS to improve overall performance while ensuring application requirements are met.

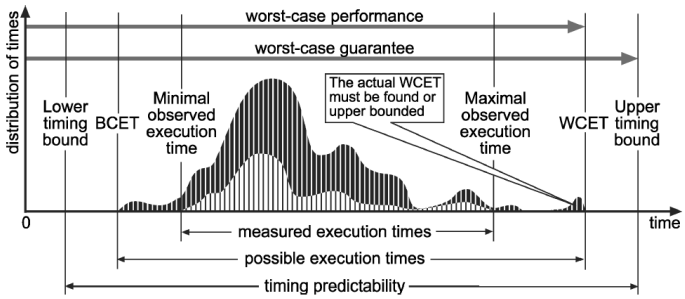


Figure 1.4: Timing analysis of a system [12]

1.1 Motivation

With the increase in multiple applications of varying requirements, constraints and criticality sharing computing and communication resources, the environments in which these applications are running are no longer static throughout their runtime but dynamic. For example, dynamic environments are seen in Fog computing platforms running Internet of Things (IoT) services [11]. Such fog nodes should support flexible provisioning of resources to applications and also services need to be strongly isolated from each other due to security concerns. Hard and soft-real-time applications sharing the same platform give rise to new sets of challenges. A fundamental research question in MCS is how to accommodate the conflicting requirements of partitioning for safety assurance and sharing for efficient resource usage [10].Traditional techniques include allocating dedicated resources to critical applications and isolating them spatially and temporally from the rest of the system. Computing and communication resources can be shared across applications with different criticality and spatial and temporal partitioning of resources are applied over time to avoid inter-application influence. Care must be taken because isolation is conservative and can result in wastage of resources.

Consider Figure 1.4, which shows basic timing analysis of real-time tasks [12]. A task usually has variations in execution times depending on the input data or environment behaviour. The set of all execution times is a large state space and the shortest and longest time is referred to as Best-Case Execution Time

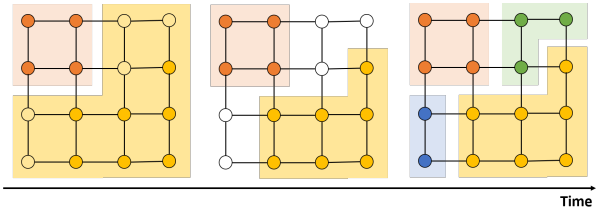


Figure 1.5: Illustrating runtime traffic isolation over time [13]

(BCET) and Worst-Case Execution Time (WCET). Since this state space can be large, in industry [12], a common method to compute execution time bounds is to measure for a subset of the possible executions—test cases resulting in minimal observed and maximal observed execution times. Since such methods in general, overestimate the BCET and underestimate the WCET, they are considered not safe for hard real-time systems [12]. This method is usually called dynamic timing analysis. Execution time bounds like WCET and BCET of a certain task can be computed by methodologies that take into consideration all possible execution times of the task. WCET bounds depict the worst-case guarantees computed by the methodology/tool. Typically, these methods use abstractions to make timing analysis feasible. Abstraction can result in loss of information and thus overestimating WCET. The degree of information loss depends on the methodologies, characteristic of software and hardware architecture.

A challenge in mixed criticality systems is integration vs isolation. Techniques like runtime resource consumption and management are proposed for effective resource utilization over time. This improves flexibility, adaptability to dynamic environments in mixed criticality systems. Success for an application here depends on its criticality. Real-time critical applications need to meet their hard deadlines while other less critical applications have softer deadlines. This results in a dynamic mixed criticality system. Communication support like the NoCs is an integral component for the efficient functioning of such a system. Hard guarantees are expected from the NoC by such real-time critical applications. A mixed critical NoC should be able to support both kinds of traffic.

A NoC comprises of routers interconnected by links and information is injected via a Network Adapter/Network Interface into the network and transmitted across a path computed by the routers. A simple NoC is shown in Figure 1.3. To illustrate dynamic environments using a simple example let us consider multiple applications of varying criticality sharing such a NoC as shown in Figure 1.5. In this example, four applications of varying criticality are sharing the network resources over time on a 4x4 mesh topology. Each router in the Figure is connected to a processing tile. In this illustration, only the routers are shown for simplicity. The performance of the NoCs depends on various factors, including choice of routing algorithms, topology, arbitration, flow control, switching techniques etc. A NoC that can support and adapt to a dynamic mixed critical environment and make changes at runtime to improve overall performance is beneficial. There is a benefit when utilizing runtime adaptations, but supporting this and in parallel maintaining guarantees is an open research problem as mentioned in [9]. Exploiting the inherent distributed nature of a NoC, proposing techniques and methodologies to make the network more adaptable and flexible while meeting varying application constraints is the focus of the thesis.

1.2 Contribution

Managing diverse applications on a shared heterogeneous system comprising of hundreds of cores can get unpredictable and fixing all parameters at design time can be conservative resulting in wastage of resources. Such a system can benefit from a runtime adaptive on-chip interconnect. Here the focus is on developing techniques that improve the adaptability of the NoC and utilize its inherent distributed nature in the context of mixed criticality systems. A NoC which is able to adapt at runtime to varying application requirements while meeting real-time constraints for critical applications. The aim is to guarantee QoS for critical applications while improving the overall performance of critical and non-critical applications.

Adaptive techniques exist at different levels of NoC in literature [9]. In this work, the focus is on exploiting the inherent distributed nature of the NoCs, and propose

increasing its adaptivity at different levels while ensuring QoS guarantees for critical applications. QoS is the overall performance comprising of latency and throughput achieved by the NoC. Guaranteeing the QoS is especially important for applications that have real-time, safety or security requirements. If multiple applications having varying criticality levels share the NoC resource, it can get challenging to meet underlying requirements. Performance on such systems can be improved if the NoC is flexible and supports provisioning and isolation of resources. An overview of the contributions is provided below.

DREAM NoC: In this work, the potential of using reinforcement learning based routing techniques to improve flexibility and performance especially targeting mixed criticality systems is investigated. A DREAM NoC: A **D**istributed **R**einforcement learning **E**nabled **A**daptive **M**ixed-Critical Network-on-Chip is presented. DREAM is a distributed and self-learning NoC that uses a topology agnostic reinforcement learning enabled routing algorithm based on the Ant Colony Optimization (ACO) metaheuristic. It comprises of runtime discovery of paths and selection of optimal routes over time based on traffic fluctuations. A hardware friendly router architecture of the reinforcement learning based routing using ACO is presented and evaluated.

DREAM Framework for spatial isolation in MCS: An approach when implementing high criticality applications traditionally is to have dedicated resources allocated, to avoid the influence of non-critical applications. Techniques like temporal and spatial partitioning of resources are commonly used in such mixed criticality systems to ensure real-time requirements are met. Here the work utilizes ACO based reinforcement learning, which is a topology agnostic self-learning NoC in aiding in the spatial isolation of NoC resources. These concepts are integrated and tested using the developed DREAM NoC, by defining specific learning and application phases. A learning phase is dedicated to building preliminary routing tables to be utilized in the subsequent application phase. This supporting DREAM framework is presented to efficiently partition NoC resources and aid flexible spatial isolation of resources.

Multi-Layered NoCs for MCS: The work also includes investigating isolation techniques using multi-layered NoCs. In this method, critical traffic and non-critical traffic are de-coupled to reduce influence over each other and maintain guarantees for critical applications. Here a hierarchical and multi-layered NoC targeting mixed critical traffic is proposed. Supporting adaptive, criticality and congestion aware routing for multi-layered topologies is presented.

Adaptive multicast techniques: Increasingly traffic with single source and multiple destinations are popular. This is commonly seen when memory coherency regions are established and supporting memory based packets are injected. The work also investigates improving traditional multicast routing by making it more scalable, adaptive and flexible by developing a Block-Based multicast routing technique.

1.3 Outline

The work is organised as follows. An overview of fundamentals relevant to the work is presented in Chapter 2. In this Chapter, basic NoC components and their functionality, an overview of routing, switching and flow control techniques are described, along with an overview of reinforcement learning concept. An introduction to the InvasIC computing platform, where the supporting *i*NoC is used as a basis for the architecture in this work is described here. Related work is discussed in Chapter 3. An overview of the selected state of the art NoCs in industry is presented along with an overview of adaptive routing techniques. Topology agnostic routing algorithms and the evolution of reinforcement learning based routing using the ACO are presented. At the end of this chapter, limitations in existing work, with a focus on the NoCs and routing techniques targeting mixed critical systems are discussed.

The proposed adaptive techniques at different levels of the NoC is shown in Chapter 4. Broadly, adaptivity is introduced at the routing and topology level. At the routing level, reinforcement learning based routing techniques is investigated, adaptive congestion and critical aware routing for multi-layered NoCs and adaptive

multicast routing are proposed. The concept of DREAM NoC, which is a self-learning NoC, is introduced . Reinforcement learning based routing techniques based on ACO is described and the potential of such a topology agnostic routing to aid in spatial isolation is highlighted. Adaptations in multi-layered NoCs are described next. Investigations done to decouple critical and non-critical applications to reduce inter-application influence is presented. Supporting congestion and criticality-aware routing is described. Concepts of Block-Based multicast routing which is adaptive and scalable to improve upon existing traditional multicast techniques are described. Chapter 5 presents the concept of the DREAM framework, which describes how the DREAM NoC can be used to aid in spatial isolation in MCS. Scalability analysis is discussed compared to traditional approaches. How the QoS guarantees are supported for critical applications is discussed.

The NoC architecture, implementing the proposed DREAM NoC and framework is presented in Chapter 6. Along with the architecture required for multicast routing, multi-layered topology and supporting algorithms. It starts with the design constraints and then the router architecture is discussed. Mathematical complexity of the reinforcement learning based algorithm and modifications done to make it hardware friendly is discussed. How deadlock and livelock avoidance are implemented is also presented. Evaluations and analysis of results are done in Chapter 7. An overview of the HDL test environment is described. Proposed topology agnostic routing using reinforcement learning is compared against state-of-the-art techniques. The novelty of the DREAM framework is highlighted here. Evaluations of congestion and criticality aware routing for layered NoCs are presented and compared. Block-Based multicast is compared against traditional techniques and the advantages in scalability, latency and throughput are discussed. Communication task graphs of test application cases are presented. Reinforcement learning enabled routing based on ACO is also extended to multi-layered NoCs and results are compared and analysed against congestion-aware adaptive routing techniques. The work is summarized in Chapter 8, with an outlook on future work.

2 Fundamentals

This chapter presents an overview of relevant fundamentals of a Network-on-Chip (NoC), of its evolution over the years and its role in mixed criticality systems.

2.1 System-on-Chip

With the trend in Moore's Law, the integration density steadily increased, which enabled complete systems to be implemented on a single piece of silicon and were referred to as a System-on-Chip (SoC). SoCs are widely used in domains like automotive sectors, telecommunications, and consumer electronics to name a few. The functional and performance requirements of an application were typically addressed by combining a custom hardware solution together with software running on a standardized embedded processor core. Where the standard cores were designed to interface with special-purpose hardware [2].

The ending of Dennard scaling in 2004 led to the integration of multiple efficient cores instead of one inefficient core. The number of processing cores on the chip increased, leading to the development of Multi-Processor System-on-Chip (MPSoC). MPSoCs comprising of mainly processing cores are also referred to as multi-core architectures.This led to switching from solely Instruction Level Parallelism (ILP) to Data-Level Parallelism(DLP) and Thread-Level Parallelism (TLP) and later also Request-Level Parallelism (RLP). The improvement in performance due to multiple processing cores depends on the degree of parallelism of the software. Amdahl's Law describes limits to the number of useful processing cores. If, for example, 10% of the task is serial and 90% is the portion that can

be parallelized, then the maximum benefit in performance (speedup) using parallelism is 10, even if one is increasing the number of cores. The overall speedup representation can be provided by equation 2.1.

$$Speedup_{overall} = \frac{ExecutionTime_{old}}{ExecutionTime_{new}} = \frac{1}{(1-Fraction_{enchanced}) + \frac{Fraction_{enchanced}}{Speedup_{enhanced}}} \quad (2.1)$$

The Speedup $S(p, n)$ due to parallel execution on n number of cores when compared against sequential execution of a program can be given by equation 2.2. Where p represents the fraction of the program that can be parallelized.

$$S(p, n) = \frac{1}{(1-p) + \frac{p}{n}} \quad (2.2)$$

With the increase in the number of SoC components, communication also grew, leading to the development of complex on-chip communication architectures.

2.2 On-Chip Communication

Digital electronics are becoming communication limited. The factor that is limiting cost, performance, power, and size is not arithmetic or control logic but the *movement of data* [14]. Traditional bus-based communication was unable to keep up and became a performance bottleneck with the increase in the number of processing cores. Busses have dedicated physical communication channels shared by multiple cores, and bandwidth is limited by bus line. With the growing number of cores, bandwidth is constant, leading to bad scalability. An illustration of a commonly used shared-medium backplane bus is shown in Figure 2.1 [15]. It is an interconnection structure for a small number of bus masters and a large number of bus slaves. The information on the bus belongs to data, address, and

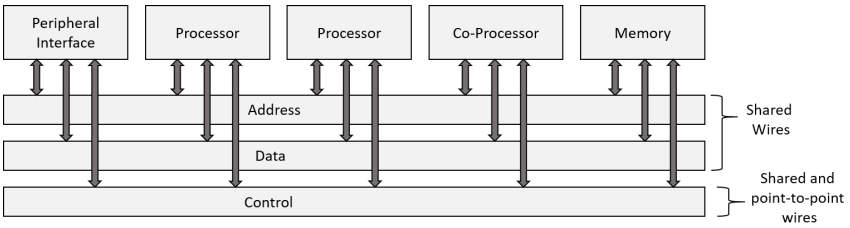


Figure 2.1: An illustration of a shared-medium backplane bus [15]

control classes. Bus arbitration is required to deal with multiple concurrent requests. An example of a bus standard widely used is (Advanced Microcontroller Bus Architecture) AMBA by ARM [16].

Conventional bus-based communication is convenient for systems with less than 5 processors and usually not more than 10 bus masters [15]. Bus-based systems are usually not scalable, and this becomes a performance bottleneck when the processors are increased. Another critical limitation when the number of processors increases is energy waste caused by functional congestion [15]. Energy considerations limit the usage of buses in large-scale systems. Other challenges which SoCs faced include synchronization of future chips with one single clock source with negligible skew. A potential solution is using multiple clocks and utilizing Globally Asynchronous and Locally Synchronous (GALS) concept. SoCs then become more distributed even on a single silicon substrate due to the absence of a single timing reference. Global control can get challenging as tracking each component state becomes necessary. In such cases, each component will tend to initiate data transfer autonomously, depending on necessity. This has led to a more fully distributed communication pattern with little to no global co-ordination. Based on these challenges, researchers postulated that the layered design of reconfigurable micro-networks can achieve efficient on-chip communication with higher bandwidth and scalability communication on SoCs [17]. Such micro-networks were inspired by the methods and tools used for general networks. Over a decade ago, NoC was proposed by several researchers. Examples include L. Benini and G. De Micheli [17], W. J. Dally [[18] and A. Jantsch [19] to name a few.

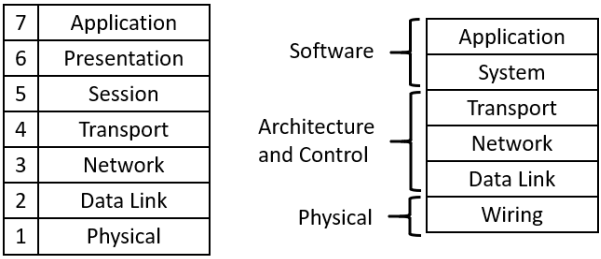


Figure 2.2: OSI Model on the left and the micronetwork stack paradigm from [17] on the right

2.3 Networks-on-Chip

With the limitations of traditional bus-based architectures, on-chip communication solutions are moving towards networks. Interconnection networks offer modular, scalable, high-bandwidth, low-latency communication between components. NoCs were developed by researchers by borrowing models and techniques from the network field and applying them to SoC designs [17]. NoCs were viewed as a micronetwork of components. Authors in [17] discussed the application of the protocol stack to interconnects, resulting in a micronetwork stack paradigm. The protocol stack/ Open Systems Interconnection (OSI) model layer and the micronetwork stack paradigm are shown in Figure 2.2. The network would support communication between such components and meet Quality-of-Service (QoS) requirements like reliability, latency, throughput, and energy bounds, for example. While keeping in mind the limitations due to intrinsically unreliable signal transmissions and delays on wires [17].

A general overview of a simple NoC interconnecting 16 nodes in a mesh topology is shown in Figure 1.3. The nodes are often referred to as tiles. Basic NoC components usually are routers, links and Network Interface (NI) / Network Adapter (NA).

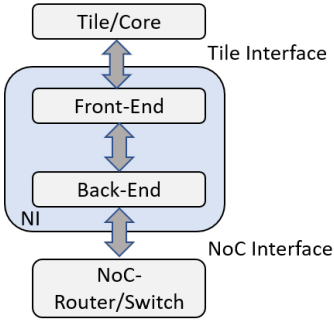


Figure 2.3: Basic structure of a Network Adapter/Network Interface [20]

2.3.1 Network Interface

Network interfaces are used to attach system components to the rest of the network. Some examples of their functionalities include packetization of data, error handling, protocol conversion and flow control. The basic structure of an NA is illustrated in Figure 2.3. An NA typically has two interfaces: Tile Interface and NoC interface. The tile interface connects the NA to the Tile, whereas the NoC interface connects the NA to the network.

2.3.2 Routers

Routers forward the data from the source to its destination node. Routers determine paths of the packet across the network based on certain routing principles and flow control. Router architectures are varied; they can be simple and lightweight to reduce resource overhead or complex depending on the application requirements. An example of a router architecture used in [20] is shown in Figure 2.4. This is also the base router used in this work to conduct investigations.

The example router architecture here is a packet switched router, which uses input buffers, routing control units and a crossbar. Input buffers store whole packets or parts of them before they are ready to be processed and transmitted across the router. When a router receives an incoming packet, it is assigned to an input

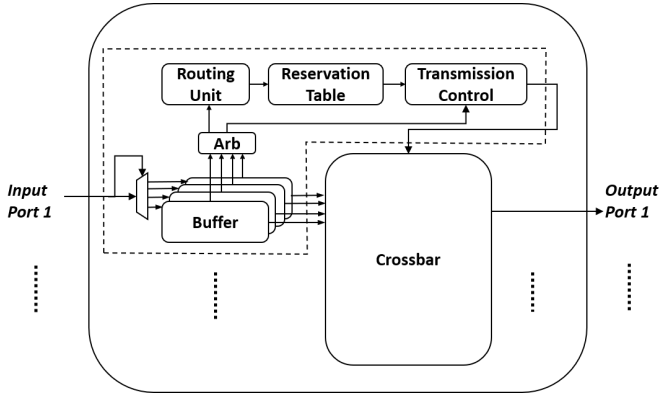


Figure 2.4: Base Router Architecture [21]

buffer. In this example, four input buffers are assigned per port. An arbiter based on Round Robin arbitration selects a buffer to read the packet out of for further processing. A routing algorithm like XY routing is implemented in the routing units. Once an output port is selected, the reservation table reserves the path in the crossbar, and transmission control handles the transfer of packets across the crossbar. The crossbar comprises of multiplexers that connect the input to the output port. Packets are forwarded to the downstream router once there are enough free spaces in the downstream router.

Certain routers can be buffer less to reduce resource overhead and reduce energy consumption [22]. Suitable routing algorithms need to be implemented then, like deflection/hot potato, since packets are not stored in the router. Packets in bufferless routing are forwarded continuously. The router always forwards packets from its input port to an output port regardless of whether the output port results in the lowest hop count. The reduction in resource overhead can be beneficial as input buffers can occupy a major portion of the router resources, but tradeoff needs to be taken into account, and simpler routing has to be implemented.

2.3.3 Links

Links are the physical connection between two nodes/routers/switches and can be realized by wires. Physical wires are a cheap commodity for networks due to the relatively small wire pitch in silicon technologies and the availability of wiring in several layers [15] [22]. Consequently, control and data are physically separated, typically resulting in a cleaner design. Due to low overhead for wiring, the links are usually implemented in a bidirectional manner with separate wires. The disadvantages are signal delay and dispersion of on-chip wires. As delay grows quadratically with wire length due to the combination of resistive and capacitive effects. Alternate link technologies are being investigated since performance bottlenecks have emerged due to bandwidth demands for multicasting and broadcasting as described in [23]. New interconnect technologies include optical interconnect and Radio Frequency (RF) based interconnects such as wireless NoC (WiNoC). In wireless NoCs, miniaturized on-chip antennas are implemented to communicate between routers/nodes.

In this thesis, the focus is on wired NoCs. Data is transferred on links and usually has a fixed bit width in such NoCs. Data is typically measured in bits, and a unit of data transferred in a single clock cycle on a link is called the phit (physical unit) [15]. When two routers synchronize each data transfer (to ensure buffers do not overflow, for example), link-level flow control is used (which can be based on handshaking). The unit of synchronization is called a flit (flow control unit). A flit is at least as large as a phit. An overview of the packet structure is shown in Figure 2.5.

2.3.4 Topologies

Nodes are connected via links to form different topologies. The choice of topologies impacts performance, implementation cost and floor planning. A measure of the bandwidth of a network is bisection bandwidth. It is the sum of the bandwidth of the minimum number of channels that need to be cut to partition a network

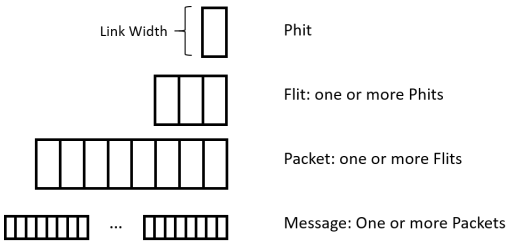


Figure 2.5: Overview of packet structure

almost in half [14]. Bisection bandwidth is the bandwidth between two parts of the network which has been cut into approximately equal parts [2].

A few examples of common network topologies are shown in Figure 2.6. Mesh is the most popular due to its uniformity and regularity. Torus has the same edge connectivity for all nodes, and it is relatively regular except for links connecting the border nodes, which are longer. Edge connectivity is the minimum number of edges to cut to disconnect a node from the rest of the network. Some systems use ring topologies, but with the increase in number of connecting nodes, the topology does not scale well. As the bisection bandwidth of the ring does not increase with the addition of more nodes. Many of these topologies are regular topologies. Examples of irregular topologies are shown in Figure 2.7. These topologies can be present in systems due to various reasons. It can be by design, due to resource constraints, result due to spatial partitioning, or can be caused at runtime due to failure in links or routers, to name a few examples. Therefore, it can be beneficial to investigate routing techniques that can support both regular and irregular topologies as well.

There has been growing interest in developing 3D topologies [24]. Examples of 3D mesh topologies are shown in Figure 2.8. When such topologies are implemented on a Printed Circuit Board (PCB), silicon layers are stacked on top of each other and the connections between layers can be using Through Silicon Via (TSV). They can be advantageous as many nodes can be compacted together with shorter hop distances between them.

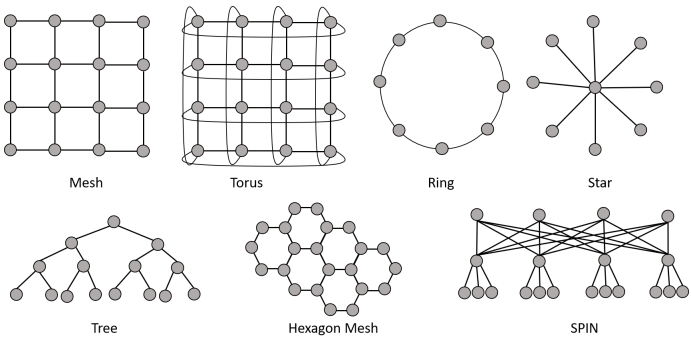


Figure 2.6: Examples of topologies

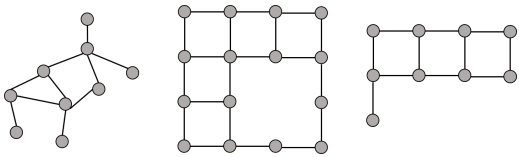


Figure 2.7: Examples of irregular topologies

2.4 Circuit and Packet Switching

Routing determines which path a packet takes in the network from a source to its destination. Switching determines how the data is transmitted. Switching can be circuit switching or packet switching.

2.4.1 Circuit Switching

In circuit switching, messages are sent in their entirety from the source node to its destination node. An end-to-end connection is established before the data can be transmitted, usually by a head flit. The head flit is injected by the source and reserves links and routers along the path to the destination. During this setup phase, relevant connections and multiplexers are configured within the routers. If the header flit arrives at the destination successfully, an acknowledgement is sent back to the source. If links are not successfully reserved, a negative

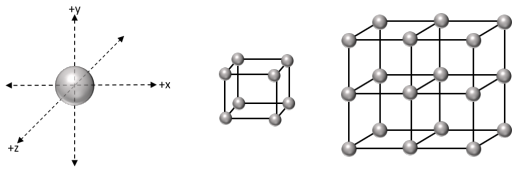


Figure 2.8: A 3D node and examples of 3D mesh topologies where (x,y,z) are 2x2x2 and 3x2x3 respectively

acknowledgement is sent. Data is then transferred by the source node when the path is reserved in an uninterrupted manner. The connection behaves as a circuit between the two nodes. Then, body flits follow the head flit along the reserved path. After data transfer, a tail flit releases the path to be used for other communication. Initial setup can have high initial latency due to routing and transmission of acknowledgement [20]. Once the setup is completed, the data transmission is efficient due to high bandwidth and low latency. Latency is kept low due to the absence of arbitration and no buffers.

Due to the reservation of path for the sole transfer of data, guarantees in latency and throughput can be determined at design time. Therefore, circuit switching can be used for real-time communication. The trade-off is that the circuit switching technique is not scalable and are limited in its flexibility. Paths are blocked for other communication during data transfer. The number of parallel circuit switched communication paths are limited by the number of available links. Circuit switching is suitable when communication between node pairs is fixed for long periods of time and when guarantees in latency and throughput are required. An early NoC example that used circuit switching is SOCBus [25] for hard real-time embedded systems. Æthereal NoC [26] uses a combination of circuit switching and packet switching, which offers best-effort communication. In the InvasIC NoC (*i*NoC) [20], hybrid routers are implemented, which support both packet and circuit switching, offering both Best Effort(BE) and Guaranteed Service(GS) communication.

2.4.2 Packet Switching

In packet switching, data is divided into packets and transmitted from source to destination. Setup of links between nodes is not required, and the packets are directly injected into the network. Packets of the same data can take different routes to reach the destination and with uneven latency. The reordering of packets needs to be handled as packets need not arrive in the same sequence at the destination as they are injected at the source node. Reordering can be handled at the router level or at the destination side. Unique packet IDs are usually assigned to aid in reordering.

Due to the absence of a setup, the communication mechanism is more flexible. Frequently changing communication node pairs can benefit from such a mechanism which does not have a setup period. Best-effort communication, when enabled by packet switching supports communication between multiple pairs in parallel without taking resource reservations into consideration. When multiple communication want to access same link/router resources, arbitration is required to ensure contentions are resolved. When contention occurs, an arbiter grants access to a certain packet and other packets need to be stored in buffers inside routers for example, before they can be granted access. Buffers are storage units implemented using registers or memory to support the temporary storing of packets. The implementation cost increases due to buffers when compared against circuit switching mechanism. Packet switching is relatively more flexible when compared to circuit switching. Links are occupied only for the duration of the packet. Links are utilized more efficiently and flexibly which can result in higher overall throughput.

Paths in the network can be used in a time multiplexed manner by multiple communication processes occurring in parallel between other nodes in the network [20]. This avoids blocking of a path by one communication. However ensuring latency and throughput guarantees at design time can get challenging due to the influence of other data packets sharing the same paths. Therefore, in general, packet switching does not support QoS of latency or throughput. When multiple packets arrive at the same router, input buffers are usually used to temporarily

store incoming packets or parts of packets before they are ready to be processed. This is to avoid dropping of packets. For buffer less routers, routing algorithms like deflection/hot potato routing need to be implemented. The method in which packets are forwarded from one router to the next can vary. An example is Store and Forward (SAF) flow control or switching method which supports a packet switching implementation.

2.4.3 Flow Control

As packets are forwarded in the network, flow control mechanisms manage resource allocation like links and buffers to packets. In SAF flow control, a packet is forwarded to the next router only if the entire packet can be stored in its buffers. The advantage is that during inter router packet transfer, transmission is not stalled after initiation as the entire packet is accepted by the receiver. A disadvantage is that a packet is not forwarded by a router until the entire packet is received, which increases the delay. Another disadvantage is the requirement of large buffers to store entire packets. Buffers in NoCs are a critical resource due to their implementation cost and power consumption [27]. Due to these disadvantages, SAF is not commonly used in NoCs. An example where they are used is Nostrum NoC [28], where the packets are one flit size to reduce the effect of the disadvantages.

A flow control mechanism that aims to reduce the latency disadvantage in SAF is Virtual Cut Through (VCT) switching. In this flow control, a router forwards the first flit of a packet to the next router as soon as there is space avaliable to store the entire packet. The router does not wait for the entire packet to be received before it starts forwarding to the next router. If there is not enough space in the next router, the entire packet is stored within its router.

The wormhole flow control scheme further reduced the buffer requirements by reducing it to one flit. In wormhole switching, a router forwards a flit of a packet as soon as it receives it. It does not wait for the reception of the entire packet and the downstream router need not have buffer space to store entire packet. This decreases the delay and buffer requirements when compared to SAF and VCT. The

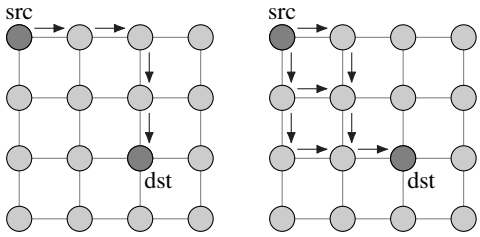
disadvantage is that links across multiple routers can be blocked due to packets strung over them, potentially causing contention. A way to ease this is by using Virtual Channels, which enable processing and forwarding of multiple packets in parallel. Due to the advantages of Wormhole switching, it is a popularly used flow control in many packet switched NoCs [15]. Some examples include Hermes[29], Æthereal[26], MANGO and *i*NoC[20].

2.4.4 Virtual Channels

Dally in [30] introduced the Virtual Channel (VC) concept to decouple buffer resources from transmission resources. This allows active messages to overtake blocked messages using bandwidth that would have been otherwise idle. Typically, a predefined number of virtual channels are built per port and the virtual channels share the physical link according to a certain scheduling strategy. The virtual channel concept can be applied to both packet and circuit switching strategy to effectively use the available bandwidth. Cost includes the complexity of the crossbars and depending on the switching scheme, buffers for each VC need to be implemented. The complexity of an N port router is $N \cdot (N-1)$. The complexity of an N port router with a V number of Virtual channels is $N \cdot (V \cdot (N-1))$.

2.5 Routing Algorithms

Path of a packet in a network is determined by a routing algorithm. One of the goals of routing algorithms is forwarding packets on paths with the shortest latency. Other beneficial features include efficient load distribution, prevention of deadlocks and livelocks. Deadlocks are situations where there is mutual blocking of links resulting in packets getting blocked and not reaching the destination. Livelocks are scenarios where packets are being forwarded in the network without reaching its destination node. Routing algorithms can be classified based on several different features like place of routing decision, minimality and adaptivity for example [31].



(a) Static XY routing **(b)** Partial adaptive routing

Figure 2.9: Static XY routing and partially adaptive routing

2.5.1 Path Adaptivity

There are different implementations of adaptivity. Here, we look at the degree of adaptivity in choosing a path from a range of possible options. In the simplest case, there is no adaptivity and all packets travelling from a source to a destination do so on the same static path as shown in Figure 2.9a. Static XY routing is a popular routing algorithm due to its simplicity of implementation and is applied only on mesh based networks. Here, the packets travel first in the X direction towards the destination and then in the Y direction towards its destination. Implementation is simple with low cost overhead. Because of their simplicity, static and deterministic routing algorithms like XY are popularly implemented in NoCs including *i*NoC[20]. An important advantage of such static algorithms is path predictability which enables ensuring guarantees in latency and throughput. Metrics like worst case latency and throughput can be computed at design time. This information is vital for critical applications to determine if real-time requirements can be met by the network. On the other hand, such static routing can suffer from hot-spots, congestion and low fault tolerance. Hot-spots occurs when a router receives a higher than average amount of traffic and can get overloaded. Certain links can be more heavily used than others leading to uneven load distribution in the network. Since links are shared by multiple packets this can have a detrimental influence on latency and throughput. If a router or link stops functioning due to a fault for example, static routing cannot route around faulty components. Thus failing to deliver packets successfully resulting in having low fault tolerance.

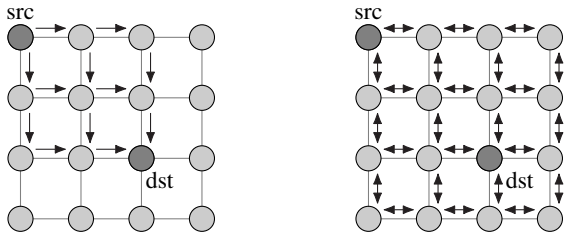
If a routing algorithm chooses a path among a subset of possible paths, the routing algorithm is typically called partially adaptive. Paths are usually excluded in order to guarantee deadlock freedom by restricting specific turns. A popular example of a deadlock-free partially adaptive routing algorithms is odd-even routing. Figure 2.9b shows the possible paths under the odd-even turn model. Note the bottleneck were all paths involve a single link west of the destination node. Lifting these restrictions leads to a fully adaptive routing algorithm, which allows all paths like in Figure 2.10. This flexibility comes at the cost of increasing routing complexity and the challenges of dealing with deadlocks.

2.5.2 Minimality

Routing algorithms can also be classified further based on their minimality. A minimal routing algorithm allows only the shortest paths between two nodes, while non-minimal ones do not have these restrictions. Applying the condition of minimality in routing algorithms ensures that packets reach their destination after a fixed number of hops. This prevents packets from travelling in the network without reaching the destination which can cause livelocks. Advantage of non-minimal routing algorithms include efficient routing of packets across non-minimal paths when minimal links are unusable either due to link failures or congestion. For example, if the nodes on minimal paths in Figure 2.10a were blocked, a minimal routing algorithm could not route packets. Alternate paths in this scenario is possible with non-minimal routing as illustrated in Figure 2.10b.

2.5.3 Place of Routing Decision

The place of routing decision is an important factor for determining complexity of routing algorithms. In source routing, all routing decisions are made at the source node in advance and are stored in the packet. The entire path is computed and stored in the packet before the injection of the packet into the network. Intermediary routers follow the path information in the packet and forward accordingly.



(a) Full adaptivity: only minimal **(b)** Full adaptivity: minimal and non-minimal

Figure 2.10: Fully adaptive routing comprising of minimal and non-minimal paths

However, only information available to the source node is considered in computing paths and the path length is limited by the avaliable packet header size, which restricts scalability. An alternate routing method is distributed routing, where each intermediate router is responsible for its local decision. This brings less protocol overhead and enables access to local information, but can make the router architecture more complex as routing decisions need to be made at each router.

2.6 Deadlocks and Livelocks

Deadlocks are caused when packets have cyclic dependencies. There is mutual blocking of network links causing the packet to get blocked and not able to be forwarded. This results in blocking of paths for other packets and thus having detrimental effect on performance. There are techniques which can be implemented in routing algorithms to avoid deadlocks. An example is XY algorithm which restricts certain turns to avoid cyclic dependency. Deadlocks can be avoided by restricting one of the turns in a turn model [14]. In the turn model, all possible ways a packet can turn in a topology are defined. An illustration of a turn model for a 2-D mesh is shown in Figure 2.11. NoCs, that employ virtual channels (VCs) like the *i*NoC, can maintain deadlock freedom by restricting turns for certain VCs [32].Livelocks are scenarios where packets travel in the network without reaching

(a) Clockwise **(b)** Anti-Clockwise

Figure 2.11: Turn Model for a 2D Mesh network

its destination. Such scenarios can be avoided by implementing a hop counter and dropping packets once a certain number of hop count is reached.

2.6.1 Deadlock Avoidance vs Deadlock Recovery

Deadlock avoidance methods include ensuring packets do not involve in deadlocks. This is done by using deadlock free routing algorithms like XY, where certain turns are restricted. Or by using bufferless routers and routing algorithms like hot potato where packets are not stored but forwarded continuously or dropped. These techniques can be restrictive and conservative which can inhibit performance.

The possibility of deadlocks occurring depends on various factors, including traffic and the type of routing mechanism. If the possibility is low, the effort into restricting turns or ensuring deadlock free routing can inhibit performance. In such cases it can be beneficial to accept the possibility of deadlock and not inhibiting the flexibility of the routing but having deadlock recovery mechanisms. Employing deadlock avoidance in routing algorithms can restrict its flexibility. In situations where probability of deadlocks are low, it is more efficient to employ deadlock recovery techniques. Here, once a deadlock occurs in a network, it is detected and resolved using different techniques. An example is [33], where a SWAP mechanism is proposed intended to break deadlocks by swapping a blocked packet with a buffered packet in the next hop.

2.7 NoCs in Real-Time Systems

Real-time applications form a significant portion of embedded applications. Complex constraints are imposed by real-time applications on NoCs. Successful real-time communication is based on correct logical results and the completion of the computations within certain time bounds. A packet arriving at the destination too late results in failure and even causes catastrophic consequences. So, it is vital that computation and communication meet certain deadlines. In Hard Real-Time Systems (HRTSs), a missed deadline is not allowed even in a worst-case communication scenario as it would cause catastrophic consequences [9]. Soft Real-Time Systems (SRTSs) have more tolerance, and allow few missed deadlines. Therefore transmission delays between nodes needs to be predictable to guarantee latency. But predictability is challenging especially for a medium which is shared by different applications running in parallel and congestions can occur. NoCs supporting real-time applications need to provide guaranteed service performance with respect to latency and throughput. Real-time applications also impose cost and energy efficiency requirements. Real-time MPSoCs are subjected to varying application specific demands during runtime and varying workloads at runtime, motivating the need to design adaptive NoCs.

2.7.1 NoCs in Mixed Criticality Systems

When components of different distinct criticality levels are integrated into the same system, they are referred to as Mixed Criticality Systems (MCS). The criticality level is the level of assurance needed against failure. Examples of criticality include safety-critical, mission-critical and non-critical [10]. This trend of mixed criticality systems is increasing when designing real-time embedded systems. These platforms can be complex single-core or multi/many-core. Many complex embedded platforms in automotive and avionics sectors are moving towards mixed-critical systems to meet stringent non-functional requirements. These non-functional requirements include cost, space, weight, heat generation and power consumption [10]. An example of a mixed criticality system is an

unmanned aerial vehicle [10]. The safety-critical flight control software in it must be certified by a civil aviation authority to be able to operate in a civilian airspace. Mission-critical software responsible for capturing and processing images need to be "fit for purpose". Software for route planning for example is desirable, improves QoS but is less critical. The criticality status of a system component determines the degree of rigour applied in the design and analysis to determine functional correctness and resource usage. Resource usage includes process execution time and communication bandwidth for example.

In complex mixed criticality MPSoC platforms, NoCs are popularly used and the networks can interconnect components of difference criticality levels. NoC's communication resources like links and routers have to be shared by applications of different criticality level. Packets belonging to varying criticality can travel on the same channels. So a fundamental challenge is how to ensure behaviour of low criticality applications does not impact the behaviour of higher criticality applications. A fundamental research question in mixed criticality systems mentioned in [10] is, how *"to reconcile the conflicting requirements of partitioning for safety assurance and sharing for efficient resource usage"*. An approach when implementing critical applications traditionally is to have dedicated resources. However, isolation can be conservative and can result in the wastage of resources.

2.7.2 Adaptive NoCs in Mixed Criticality Systems

Real-time MPSoCs are subjected to varying application specific demands during runtime and varying workloads at runtime, motivating the need to design adaptive NoCs. With the increase of applications in mixed critical systems, care must be taken between integration to save resources and isolation to ensure deadlines are met. The focus of this thesis is on investigating and developing effective adaptive NoCs for mixed criticality systems. Performance on such systems can be improved if the NoC is adaptive and supports provisioning and isolation of resources at runtime. An aim is to ensure QoS for critical applications is guaranteed while improving the overall performance of critical and non-critical applications.

Adaptive schemes enable the NoC to support real-time applications in a flexible manner that have varying demands at runtime. Characteristics of the NoC are dynamically changed to meet application requirements while efficiently utilizing resources. Adaptations in NoCs can be at different areas of the communication architecture like switching techniques, resource allocation or routing schemes to name a few. An example is a re-routing technique described in [34], here the NoC reacts to changing network load conditions and reallocates established paths if alternate paths are available. In this thesis, we investigate and propose multiple adaptive techniques at the routing and topology levels. On the routing level, we investigate reinforcement learning based routing to develop topology agnostic routing mechanisms for mixed-critical NoCs. An overview of the reinforcement learning concept is described next.

2.8 Reinforcement Learning

One of the basic ways to acquire knowledge is through interacting with the environment. Over the years, researchers have investigated computational approaches to learning from interaction. Reinforcement Learning (RL) is one such approach, it is more goal-directed learning from interaction compared to other approaches to machine learning, as explained in [35]. Reinforcement learning problems involve learning what actions to take to maximize a certain reward signal. They are closed-loop problems because the actions of the learning system influence its later inputs. The learner is not informed which actions to choose but instead must discover which actions yield the most reward by trying them out. In challenging cases, actions may affect not only the immediate reward but also subsequent rewards. The three important distinguishing features of reinforcement learning problems as mentioned in [35] are as follows: being closed-loop, not having direct instructions as to the type of action to take and consequences of actions occuring over an extended period of time.

Authors in [35] consider reinforcement learning to be a third machine learning paradigm, alongside supervised learning, unsupervised learning, and perhaps

other paradigms as well. One of the challenges in reinforcement learning, which is different from other forms of learning, is the trade-off between exploration and exploitation as mentioned in [35], and is briefly explained next.

2.8.1 Exploration vs Exploitation

To increase reward, a reinforcement learning agent must prefer choosing actions that it has attempted in the past and had discovered that the action had resulted in producing a reward. However, to discover such actions, new actions that have not been attempted before must be tried. Therefore, in order to obtain rewards, the agent has to exploit the information it already knows, but it also has to explore new actions to become aware of better actions for the future. The agent then faces a conundrum, exploration or exploitation? And neither can be pursued solely without failure. The agent must attempt a variety of actions and, over time prefer actions that appear to be the best.

2.8.2 Reinforcement Learning System

Apart from agents and environment, the other elements of a reinforcement learning system are policy, a reward signal, a value function, and a model of the environment (optional) [35]. The policy defines the behaviour of an agent at a given time, and it can be a simple function or a look-up table, for example. A goal is defined by a certain reward signal.At every time step, the environment transmits to the agent a reward (a single number, for example). The agent's target is to maximize the total reward it obtains over a long period of time. The reward sent to the agent at any time depends on the agent's current action and the current state of the agent's environment. A reward signal indicates what is good in an immediate sense, and a value function specifies what is good in the long run.

2.9 Ant Colony Optimization metaheuristic (ACO)

Ant Colony Optimization metaheuristic (ACO) is a multi-agent framework for combinatorial optimization [36]. Here, the set of components includes a set of ant-like agents, use of memory and stochastic decisions, and collective and distributed learning strategies. ACO is strictly related to other frameworks like dynamic programming, Markov and non-Markov decision processes, and *reinforcement learning*. In this work, the implementation of reinforcement learning concepts based on ACO in NoC routing mechanisms is presented. Preliminary investigations into the potential of developing NoC routing techniques based on ACO were done as part of student's work [37].

2.9.1 ACO: Inspiration From Nature

Insects like ants evolved to find the shortest paths through difficult environments [38] Ant Colony Optimization(ACO) is inspired by the path-finding capabilities of such insects. Ants live in colonies and the number in a colony can range from hundreds to even millions. Ants communicate with each other directly via touch and indirectly through chemicals called as pheromones. All activities are controlled de-centrally through these direct and indirect communication. Self-organized finding of food and its transportation by ants is a well studied case of path-finding [39].

Ants responsible for foraging, search the colony's territory in an essentially random fashion. Along the way, they memorize their path by tracking the sun and measuring the distance with a biological pedometer [39]. Upon discovering a piece of food and feeding on it, the successful ant returns to its nest remembering the navigational information from before. The path back to the nest is marked with a pheromone. At the nest, the food item is advertised to other foraging ants. Attracted to the corresponding pheromone trail, some of them find the food and deposit the pheromone on their return path thereby repeating the process. Other

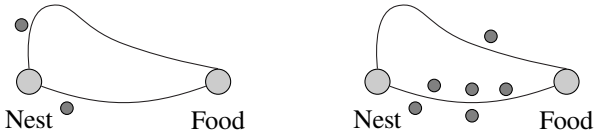


Figure 2.12: Simple example of path finding by ants travelling between Nest and Food

explorers and foragers deviating from the marked trail can create additional paths. Some paths can be sub-optimal. Ants taking a shorter path return faster, and consequently, more ants are attracted to such a path, which makes the trail even stronger with positive feedback. Because of this strong feedback loop, the colony tends to strongly favour one particular path. A simple illustration is shown in Figure 2.12. Pheromones are volatile and have a limited persistence. Hence, if a trail gets blocked or the food source runs out, no ants will be attracted anymore [38].

Ant societies feature, among other things, the autonomy of the individual, fully distributed control, fault-tolerance, direct and environment-mediated communication. There is an emergence of complex behaviour with respect to the repertoire of the single ant, collective and cooperative strategies, and self-organization. A popular research direction in ant algorithms is dedicated to the application of combinatorial optimization problems like the ACO. The shortest path behaviour of foraging ant colonies is at the very root of ACO's design. This form of distributed control based on indirect communication among agents that locally modify the environment and react to these modifications is called stigmergy [40]. The shortest path behaviour of ant colonies can be immediately recognized as an instance of a convergent behaviour. One of the advantages of ACO based routing is it multipath routing. Multipath routing strategies can aid in optimized utilization of network resources, and automatic load balancing, provide increased robustness to failures, and aid in congestion recovery. These advantages result in better throughput and end-to-end delay under heavy traffic loads. A major application of ACO is adaptive routing in telecommunication networks, as presented by Di Caro et al. with AntNet [41] [36]. The design, implementation, and testing of ACO for the problems of adaptive routing in telecommunication networks is presented by

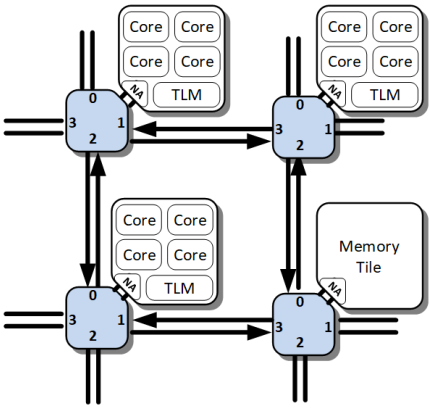


Figure 2.13: An illustration of a section of *i*NoC

Di Caro in [36]. ACO based routing algorithms for NoC descend from AntNet, which forms the theoretical basis.

2.10 Invasive Computing (InvasIC)

The work presented in this thesis was developed with the motivation to provide solutions for challenges in general mixed criticality NoCs. It was also intended to provide solutions for the communication infrastructure in the research area of Invasive Computing paradigm which investigates the design and resource-aware programming of parallel computing systems. Invasive Computing (InvasIC) is researched in a Transregional Collaborative Research Center funded by the Deutsche Forschungsgemeinschaft (DFG) [42]. In this section, we provide a brief overview of the concept and the supporting communication architecture which is experimented with in this thesis.

Since the number of processors is continuously increasing, programming on such platforms can be challenging. The main idea of Invasive Computing is to introduce resource-aware programming. A certain program receives the ability to

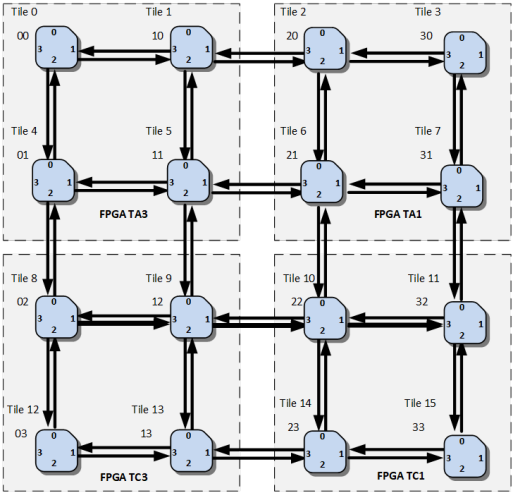


Figure 2.14: An illustration of 4x4 *i*NoC interconnecting 16 Tiles in a mesh topology.

explore and dynamically distribute its computations to its neighbouring processors while taking hardware status into account. Invasive computing [43] provided explicit handles to the programmer to specify its desired resource requirements in different phases of execution. The phases of execution are defined as invade, infect, and retreat. In the invade phase, the operating system allocates processor, memory and communication resources to be claimed by an application. In the infect phase, the parallel workload is spread across the claimed resources and executed. In the retreat phase, resources are freed from the claim if the degree of parallelism is lower or if the program terminates. Now, the program may resume execution in reduced parallelism or sequential processing commences. To support this paradigm, new programming concepts, languages, compilers and operating systems were developed along with novel MPSoC architectures. The architecture includes dynamic processors, a supporting NoC and memory hierarchy.

2.10.1 Invasive Network on Chip: *i*NoC

The invasive NoC (*i*NoC) connects the tiles in the InvasIC architecture. An overview of a section of the *i*NoC interconnecting the tiles is shown in Figure 2.13. Processor Tiles can comprise of 4-5 LEON3 processor cores, local caches and Tile Local Memory (TLM) that are connected via a shared bus. Memory tiles can be dedicated to only memory components eg. global off-chip memory.

The *i*NoC comprises of hybrid routers that support Guaranteed Service communication to support critical traffic and Best-Effort communication for non-critical traffic. An overview of the router architecture is described in Section 2.3.2 The *i*NoC supports predictable execution by enabling reservation of bandwidth. Time Division Multiplexed (TDM) timeslots on the physical link can be reserved using special Service Level (SL) bits in the packet header to reserve bandwidth. This guarantees throughput and latency on this connection.

The InvasIC architecture was prototyped on a system by ProDesign called proFPGA which consisted of four Virtex-7 2000T FPGAs. For demonstration purposes, 16 tiles were implemented across the 4 FPGAs interconnected by the *i*NoC in a mesh topology. An illustration of the layout of *i*NoC routers and links across the 4 FPGAs is shown in Figure 2.14. In this work, the *i*NoC architecture was used as a base to develop routing and topology-level adaptations. Due to the NoC size on the prototyped platform, routing level adaptations proposed in this work has been tested using a 4x4 NoC as a proof of concept.

2.11 Summary

In this chapter, an overview of NoC fundamentals relevant to the work presented in this thesis was discussed. Functioning and architecture of basic NoC components were presented along with different routing, switching and flow control schemes. An overview of the reinforcement learning concept and ACO used to develop adaptive routing mechanisms for mixed criticality NoCs was discussed. The NoC

architecture used as a basis in this work, on which routing level and topology level adaptations have been proposed and integrated, was presented.

3 State of the Art

An overview of examples of adaptive NoCs concepts, commercial NoCs, and real-time NoCs in literature are discussed in this chapter.

3.1 State of the Art NoCs: An Overview

3.1.1 TILE64

TILE64 architecture [44] from Tilera, introduced in 2007, is the first commercial NoC-based many-core architecture that was available on silicon. Traditional bus-based systems do not scale well with the increase in the number of processors. To move away from bus-based communication, authors in [44] presented a Tile Processor architecture and a supporting on-chip interconnect network named iMesh. The Tile Processor was a tiled multicore architecture developed by Tilera and was inspired by MIT's Raw processor [45]. The tiled multicore architecture is a Multiple-Instruction, Multiple-Data (MIMD) machine comprising of a 2D grid of homogeneous, general-purpose computing elements, called cores or tiles. The cores were connected using five, 2D mesh networks called as iMesh. Each network layer in the iMesh was designed for a specialized use. The first implementation of the Tile Processor Architecture is TILE64 and is shown in Figure. 3.1. It is a 64-core processor implemented in 90-nm technology and clocks at speeds up to 1 GHz. The iMesh comprised of 4 dynamic networks and one static network. The dynamic networks were the User Dynamic Network (UDN), I/O Dynamic Network (IDN), Memory Dynamic Network (MDN) and Tile Dynamic Network (TDN). The static network was named STN. Each network element connected

north, south, east, west, and to the processor and each link consisted of two 32-bit-wide unidirectional links. This enabled traffic to flow in both directions at once.

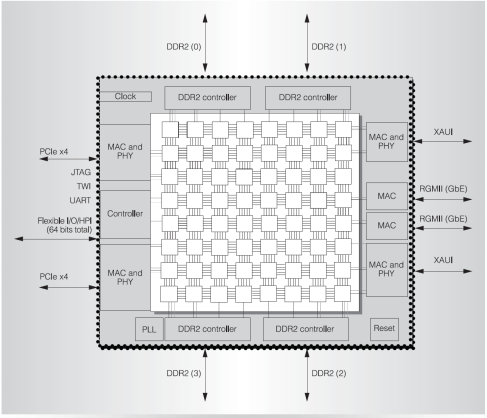


Figure 3.1: Block diagram of TILE64 processor [44]

The dynamic networks provided a packeted fire-and-forget interface, and dimension-ordered wormhole-routed. The static network was not designed to support a packetized format but supported static routing configurations at each switch. This was designed to enable streaming of data between tiles by establishing a route between them using STN network layer. The network supported circuit switched communication, having low-latency and high-bandwidth, thus suitable for streaming data. The UDN network layer was designed to serve user processes or threads. Thus enabling them to communicate flexibly and dynamically, and with low latency. The architecture uses the IDN network layer to directly communicate with the I/O devices and support system-level communications . I/O devices connected to the networks similar to the tiles. MDN network layer is used by the caches in each tile to communicate with off-chip DRAM. TDN network layer works jointly with the MDN. The architecture supports direct tile-to-tile cache transfers. The request is transmitted via the TDN and responses are transmitted via the MDN. To prevent deadlock in the memory protocol, tile-to-tile requests do not go travel in the

MDN. The design choice to have 5 independent network layers was due to the low costs of network wiring between tiles. Multilayer fabrication processes provided a plethora of wiring as long as that wiring is nearest-neighbour and stayed on chip TILE64 was followed by TILE-Gx and a version of the architecture TILE-Gx100 [46] comprised of 100 tiles arranged in a 10x10 array. The network uses a mesh topology and static dimension-order routing. Investigations into adaptive routing techniques is not conducted here.

3.1.2 Xilinx Versal ACAP and NoC

In recent years, Xilinx in 2019 introduced Versal, the first Adaptive Compute Acceleration Platform (ACAP) [47] [3]. Due to the impact of changing technological trends which resulted in slowing of processor performance as illustrated in Figure. 1.1, computing elements are increasingly becoming parallel [3]. Semiconductor industry started to exploring alternate domain-specific architectures which include vector based processing (DSPs, GPUs) and fully parallel programmable hardware (FPGAs). The Versal ACAP [3], is a software-programmable, heterogeneous compute platform which combines Scalar Engines, Adaptable Engines, and Intelligent Engines. An overview of the architecture is shown in Figure. 3.2a Scalar processing elements include CPUs which are efficient at implementing complex algorithms but are limited in performance scaling. Vector processing elements include DSPs, GPUs which are effecient in executing parallelizable compute functions. But can experience latency and efficiency penalties caused due to inflexible memory hierarchy. Programmable logic like the FPGAs can be customized precisely to execute a certain compute function, thus making them suitable at latency critical real-time applications. Disadvantage of such platforms include changes can take many hours to compile The Versal ACAP aimed to exploit the benefit of all three platforms, by combining the advantages of all three. The vector and scalar processing elements are tightly coupled to Programmable Logic (PL) and all connected using a high-bandwidth NoC. The Versal NoC [48] provides memory-mapped access to the three various processing elements.

An overview of the block diagram of Versal NoC [49] is shown in Figure. 3.2b. The Versal adaptive SoC programmable NoC is an AXI-interconnecting network. The NoC can be logically configured to complex topologies using a series of horizontal and vertical paths and customizable architectural components. The NoC comprises of a series of interconnected horizontal (HNoC) and vertical (VNoC) paths as shown in the Figure. These paths are supported by customizable hardware components which can be configured to meet requirements of design timing, speed and logic utilization. HNoC and VNoC are dedicated paths connecting integrated blocks between processor system and PL and provides high bandwidths. The Versal NoC provides QoS to manage different transactions effectively and to support latency and bandwidth requirements of different traffic streams.

As shown in Figure. 3.2b, the NoC is an interconnection of NoC master units (NMUs), NoC slave units (NSUs), and NoC packet switches (NPSs). Each of these instances is controlled and programmed from a NoC programming interface (NPI). The information in the NoC is statically routed, and these paths are determined at the design time by AMD Vivado software. The assignment of NoC ingress and egress points to specific NMUs and NSU and the routing paths between ingress and egress are computed at design time by the NoC compiler which is part of the AMD Vivado Design Suite. The NoC compiler considers connectivity of the design and the QoS constraints of the designer to provide a globally optimal solution.

3.1.3 Æthereal

Authors in [26] presented the Æthereal NoC, which focused on providing Guaranteed Service (GS) communication. The NoC was developed in Philips Research Laboratories and included the hardware, a supporting programming model, and a design flow. The NoC comprises of routers having combined GS-BE router model, provided both GS service for critical applications, and BE service for non-critical communication. GS services are provided by using the concept of contention-free routing or Time-Division-Multiplexed (TDM) circuit switching.

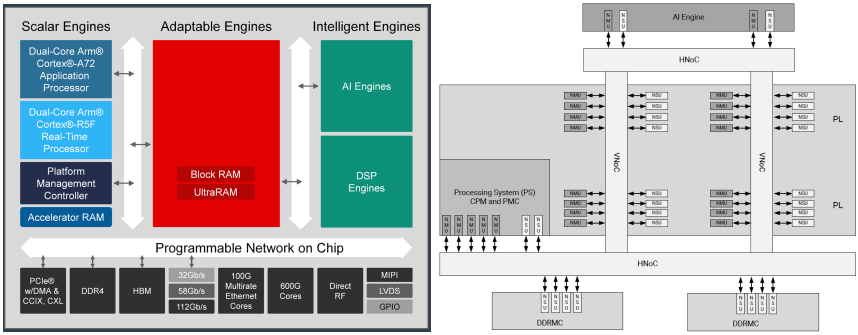


Figure 3.2: Versal ACAP (left)[47] and overview of Versal NoC (right)[49]

In the contention-free routing concept, all routers are occupying the same time slot of fixed duration. BE communication uses wormhole routing and input buffers to transmit packets. BE service uses source routing where packet header contains the entire path from source to destination. Both BE and GS flits travel on same links and BE flits can use the links only if there is no GS blocking it. The GS-BE routers are programmable with slot allocations which are generated for applications specified at design time. Two programming models are presented: centralized and distributed. Global network knowledge is required for efficient time slot allocation for all circuit-switched connections, and this is possible in the centralized scheme. However this limits scalability when allocating connection to time slots at runtime and limits flexibility when done at design time [31].

3.2 Interconnects in Chiplet based SoCs

In recent years, due to slowing transistor scaling combined with the need for high computational throughput hardware, package level integration using multi-chip-module (MCM) integration to build large scale systems are increasing in popularity. MCM is a promising approach to build large-scale system by combining multiple smaller building blocks known as chiplets into a larger system.

This approach when compared to a large monolithic die, reduces fabrication and design costs. With the increase in popularity of chiplet based SoC design, the NoC on such system can face design challenges. One such challenge include deadlocks, expecially when each chiplet is designed in isolation. Authors in [7] propose a modular routing design for chiplet-based SoCs. Here the chiplets and interposers can design its own network topology and routing in isolation and the proposed methodology restrictions on traffic flowing in and out of the chiplet to avoid deadlocks.

Authors in [6] investigated benefits and costs of using MCM with fine-grained chiplets for a deep learning inference application which requires large computational and on-chip storage requirements. Deep Learning Neural Networks (DNNs) are increasingly being used in a wide range of applications and the high compute and storage requirements motivate researchers to investigate large scale computation capable hardware. To address this, researchers in NVIDIA and Stanford in [6] present Simba , a scalable deep-learning inference accelerator using multi-chip-module-based integration. Investigations into latency variability across chiplets caused due to non-uniform latency and bandwidth for on-chip and on-package communication is conducted. To address these issues, authors propose non-uniform work partitioning to balance compute with communication latency, communication aware data placement to reduce inter-chiplet traffic and cross-layer pipelining to improve resource utilization. A prototype of Simba was fabricated, it comprised of 36 chiplets connected in a mesh network in an MCM. An overview of the architecture is shown in Figure. 3.3

Simba implements a hierarchical interconnect to connect the Processing Elements (PE) effeciently. The hierarchical NoC comprises of a NoC interconnecting PEs within a chiplet and a Network-on-Package (NoP) which connects chiplets on the same package. Both the network layers NoC and NoP uses a mesh topology and a hybrid wormhole/cut-through flow control. The system comprises of regular PEs and Global PEs which act as second-level storage for activation data to be processed by the PEs. Global PEs can transmit unicast data to one PE or multicast packets to multiple PEs local or across chiplet boundaries. A regular PE does does

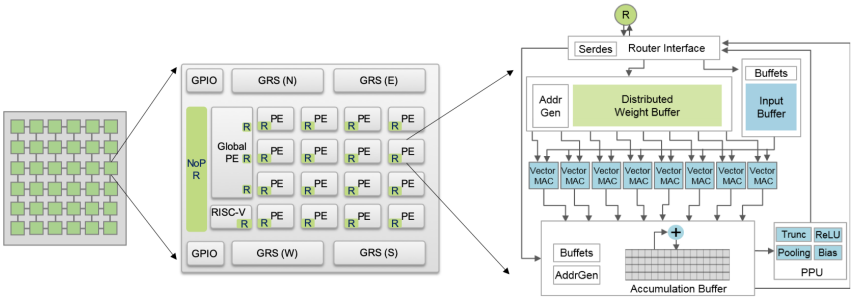


Figure 3.3: Simba architecture (Left to Right) : Simba package, Simba chiplet and Simba Processing Element [6]

not have multicast support as its computation in the system requires only point-to-point communication. Unicast communication is via wormhole flow control for large packet size and multicast communication is via cut-through to avoid wormhole deadlocks. The routing tables in NoC and NoP forward packets using dimension-ordered X-Y routing for all inter-chiplet communication. Investigating flexibility and run-time adaptivity in routing algorithms are not investigated here.

3.3 Adaptive Routing Algorithms

In this section, we focus on adaptive routing algorithms in NoCs. An overview of different concepts and approach to NoC routing in literature is presented here. If multiple paths to reach a destination node from a certain source node are available, adaptive routing algorithms potentially can choose a path among them based on network information. Quality of decision can depend on the information gathered. Some commonly used information to indicate traffic status are link utilizations and router buffer levels. For example, the *i*NoC router [31] uses Virtual Channels and link utilization for its adaptive odd-even routing mechanism. Other works like [50] and [51] use link bandwidth and distance [50]. Authors use fuzzy logic in [52] to balance traffic and power consumption. When traffic patterns are predicted

by utilizing past observations, routing decisions can be less susceptible to varying conditions in the network as demonstrated in [53] [54] .

When routers are making decisions based on only its local information, decisions made by routers far away may not be considered in the decision making and can cause issues like congestion. If routers have access to network information from a larger area, potentially better routing decisions can be made. A way to achieve this is by identifying and distributing relevant regional network information. Authors in [55] proposes an approach which introduces connections to all nodes two hops away. A control network is proposed by authors in [56], which aggregates information over longer paths. In [57] and [58] the traffic conditions of all nodes is collected in a central unit.

Implementing a dedicated network for information distribution can be useful for the routing algorithms to rely on to make suitable decisions. It is beneficial if such networks are lighter and consume lesser resources than the main network, but this can limit the network's capabilities. Otherwise if the complexity of such networks increases, it is better to utilize the existing main network to transport relevant network information.An example is shown in FreeRider [59] which presents non-local adaptive routing techniques. in this work, congestion information of distant links are transmitted in the header of existing packets, for the routers to make better routing decisions. The work in [60] utilizes the wormhole switching scheme in a unique way to gather local network information. A router sends its header flit to all of its neighbours in parallel.The neighbouring routers responds with a packet describing network status from its perspective The information is utilized to determine a suitable direction and rest of the packet flits are transmitted to the router accordingly.

3.3.1 Reinforcement Learning

Authors in [61] use reinforcement learning for data packet path evaluation. They combine local information with long-term experience about the network conditions . Routers receiving the data packets reply with a learning packet to reinforce

good decisions made at the upstream router [61, 62]. This approach is improved further in [62] by distributing the learning information in the downstream direction as well. Authors use a clustered approach to improve scalability in [63]. In [64], this strategy is further extended to non-minimal and fully adaptive routing. There are numerous heuristic optimization strategies that take inspiration from processes in nature. An example is the routing algorithm for a mixed criticality NoC in [65] which takes inspiration from the formation of synapses in the brain. In this model, connections are not just established and maintained but new ones are found in case of failure. There are similarities between ACO based routing and the strategies mentioned in this section but also have important distinctions. In ACO, dedicated packets probe and explore the network, and data packets use this information to make good decisions. This allows for exploration of paths that may be considered futile but can become viable in the future due to the change in traffic conditions. While many other routing algorithms tend to always chose the best option, AntNet distributes data packets randomly across all good routes. Causing the data traffic to spread out, aiding in congestion prevention.

3.3.2 Multi-Cast Routing

Multicast routing (one-to-many communication) support in NoCs is beneficial for a wide range of applications. This includes coherency protocols in distributed shared-memory systems, neural network implementations and fault-tolerant applications which uses redundant hardware components to name a few. In this section, we present a brief overview of existing multicast techniques and highlight differences to the proposed dynamic and scalable multicast technique described in Chapter 4.

Multicast communication can be implemented by using multiple unicast messages to reach all destinations individually. Other multicast methods can be broadly categorized into path based and tree based techniques [66]. In path based, a packet travels from one destination to another in a sequence using a single path [67]. In tree based routing, a packet follows a common path as far as possible [68]

[69] [70]. Authors in [70] describe Virtual Circuit Tree Multicasting. In this technique, a virtual tree is constructed incrementally using multiple unicast packets. Routing information is stored in Virtual Circuit Tables at each router. This technique reduces overhead of encoding multiple destinations in a multicast message. Disadvantages of this technique include virtual tree construction overhead which involves sending unicast packets using Dimension Order Routing (DoR) and presence of storage units in each router. Every time the source node changes, a new tree has to be built.

The proposed Block-Based multicast technique described in Chapter 4, has the advantage of scalable address encoding and is flexible when the positions of source and destination nodes change at runtime. Authors in [71] describe bLBDR (Logic Based Distributed Routing) which addresses routing for irregular topologies. It defines regions in the NoC using Connectivity bits and restricts the traffic to these regions. These regions are predefined based on a set of applications. Bits have to be configured and it is an overhead. Dynamically changing these regions is not discussed in the paper. Authors in [72] discuss a load-aware broadcast scheme. Tasks are mapped onto rectangular regions to constrain broadcast traffic within a region. Isolation between regions is similar to bLBDR. At most 16 regions are present and a 16 bit register is present at each output port. Two routing algorithms are used depending on the traffic load. Dynamically changing the regions is not discussed in the paper. Also the source must reside in the destination region. In our proposed solution, the source can be inside or outside the region. In [68] authors present a multicast scheme which switches between tree based and path based multicast technique. Address encoding used is a bit vector whose size is the total number of nodes in the network. Registers of the same size are present at the output port of each router. Authors in [73] present a way to decrease overall link utilization by recursively partitioning the network and then computing paths to the destination nodes. Bit string is used for destination address encoding, and whose length is the total number of nodes in the network. In [74], authors discuss a tree based routing scheme with bit string encoding for a reachable network limited to 16 nodes. Our proposed solution scales better as the bit vector is restricted to a block of nodes and is more flexible as bit vector length can change dynamically

at runtime. In [75] a data collection mechanism is introduced for region based data collection. Size of the regions are dynamically changed. The difference to our proposed technique is that in our solution, the source node can be within the region or outside and offers support when only some of the nodes in the region are destinations.

3.4 Topology Level Adaptations

In this section, we focus on multi-layer hierarchical topologies in literature. Though multi-layered topologies have been in existence for NoCs, we focus on NoCs which address improving access at runtime between frequently communicating tiles using links with reduced hop count. A hierarchical NoC is presented in [76] which aims to improve access between communication centric tiles like memory and IO tiles. To achieve this, the authors proposed a hierarchical network built from different types of subnetworks. The topologies here were chosen according to the applications requirements at design time and routing was done using look-up tables in the routers and the performance was compared against 2D mesh. The location of communication centric tiles here should be known at design time in order to choose a suitable topology. In [77], authors presented a Skip-Links architecture which dynamically alters the NoC topology at runtime to reduce logical distance between frequently communicating nodes. The aim here was to bypass routers to reduce the logical distance. The performance was compared against mesh demonstrating hop count and energy reductions. Certain restrictions are present on the placement of such Skip-Link to decrease complexity in routing.

Some of the NoCs addressing challenges in mixed criticality systems by providing isolation between different criticality packets are as follows. A runtime configurable NoC design enabling throughput guarantees with reduced impact on latency for Best Effort (BE) traffic is presented in [78]. Here the BE traffic is prioritized over the guaranteed throughput (GT) traffic in NoC routers and only switch priorities when required providing sufficient performance isolation among

different criticality levels. Authors in [79] present time-triggered communication that offers safety by establishing temporal and spatial segregation of the communication channels. The injection time of the messages are adapted based on the real execution time of computational tasks. In [80], authors propose a QoSinNoC framework, a set of QoS-aware NoC architectures. It allows non-critical communication to pass through the critical region, provided they do not utilize common router resources. Authors in [81] proposed a NoC router to support mixed criticality systems based on an accurate WCCT analysis for high-critical flows. The proposed router jointly uses Wormhole and Store And Forward switching modes for low- and high-critical flows, respectively. The assumption here is that small packets compose the high-critical traffic and the Store And Forward switching mode is used for high-critical flows.

3.5 Topology Agnostic Routing

A commonly used NoC topology is mesh due to its ease of implementation and regularity. When concurrent applications are running on a mixed criticality system, commonly used techniques like spatial partitioning to avoid inter-application influence can result in regions of irregular topologies. In addition, as described in the previous section multi-layered hierarchical topologies are increasingly being utilized. NoC topologies can be regular or irregular and can require minimal or minimal paths to route packets. Therefore it is desirable to investigate topology agnostic routing mechanisms. There exist numerous topology agnostic routing algorithms as surveyed by the authors in [82]. They include Up*/Down* (UD) and Segment-based (SR) Routing among others. As mentioned by the authors, most of them are deterministic and some optionally permit distributed adaptive routing. An overview of selected topology agnostic routing algorithms in literature is described here. Preliminary investigations into topology agnostic routing mechanisms were done as part of student's work [83].

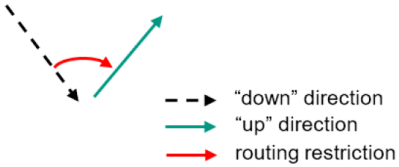


Figure 3.4: UD Restriction [83]

3.5.1 Table Based Topology Agnostic Routing

Up*/Down* Routing (UD) routing algorithm is a table-based and deadlock-free routing algorithm and was used in Network of Workstations (NOWs) interconnections like Autonet [84], Myrinet [85], Servernet [86]. The tables are computed for each specific topology. The procedure includes adding directions, adding routing restrictions and building the routing table. As the name implies, the direction "up" or "down" is added to every link in the network. There are two methodologies to add directions: Breadth First Search (BFS) spanning tree as described in Autonet [84] and Depth First Search (DFS) spanning tree as described in [87]. The first step is choosing a root node. For example, in Myrinet [85], the node with the lowest average distance to the rest of the nodes is selected as the root and rest of the nodes are considered leaves . Starting from the root, the assignment of direction to links is performed using BFS strategy. The second method to assign directions is the DFS spanning tree, which computes directions based on the former BFS spanning tree. It is suitable for complex topologies. In our work, we investigate mostly on mesh based topologies. Due to this it is simpler to utilize the BFS spanning tree for comparison purposes against the proposed schemes.

A brief overview on how deadlocks can be prevented in such schemes is presented next. After adding directions at every link ("up" or "down"), packets can potentially travel in cycles in the network causing deadlocks. To prevent this, routing restrictions are applied. An Up*/Down* rule, in which a certain routing path is strictly forbidden is illustrated in Figure 3.4 [84]. This guarantees that there is no loop to be formed in the network.

3.5.2 Non-Table Based Topology Agnostic Routing

An approach which is scalable, does not use routing tables and thus consuming less resource was presented by authors in [88] using Logic Based Distributed Routing (LBDR). Special bits called as connectivity bits and routing bits are used to define regions in the NoC and provide routing path information within this region. Packets are forwarded within the region using routing bits defined by a pre-selected topology agnostic routing algorithm. The tradeoff here is that only 30% coverage of irregular topologies is supported [89]. Non-minimal topologies are not supported by LBDR. To provide support for isolation, adaptive resource allocation for IOT applications, authors in [11] proposed pLBDR. The authors investigated utilizing LBDR for Space Division Multiplexing (SDM) NoC. A global routing algorithm is used and selective masking of routing restrictions is implemented to maintain intra-partition routing. The limitations on the partition shapes to restrict non-minimal routing exist in pLBDR as well. The coverage here is 40% of irregular topologies.

3.6 Reinforcement Learning in Routing Algorithms

3.6.1 Routing based on Ant Colony Optimization Metaheuristic

ACO based routing for NoC was presented in [90]. The routing decisions were restricted to a minimal, partially adaptive subset and guaranteed freedom from deadlocks. The results show improved latency and more consistent power consumption under synthetic transpose traffic in a mesh network when compared against other static and partially adaptive routing algorithms using a NoC simulator. The quantization of pheromone values was set to 2 bits. Authors in [91] address the scalability issues arising due to the presence of routing tables at every router and proposed modifications to the ACO approach. To avoid the growing

size of routing tables with the size of the network, an observation window is introduced. Only nodes within this window area have a space in the routing table and are explored with ant packets. The simulation results show that a small window of two or three hops has little performance degradation. This is for a mesh network with transpose traffic. Additionally, pheromone quantization is examined via simulations and authors present that 4 bits are sufficient as higher precision results in diminishing returns. A general overview of several modified versions is presented in [92]. In [93], authors present an extension to fully adaptive routing. Instead of incorporating deadlock avoidance, deadlock recovery is more focused. Deadlocks are resolved by routing deadlocked packets on a different path.Due to the pheromone based nature of routing, future deadlock possibility is now reduced with the usage of a repellent pheromone. In [94], network faults are treated similarly, where new paths are found dynamically. Details of simplifications of AntNet is not mentioned in the publications. For example, there exists distinction between "training phase" and "normal phase" in [91] and [92]. This may suggest that ant packets are not concurrently handeled or it could also be a limitation of the simulator. Additionally, there is not much information on the reinforcement strategies used. In [90], hints at adaptive reinforcement is present, but it is unclear how the reinforcement factor is calculated. Several publications do not specify their reinforcement strategy.

To investigate how QoS can be guaranteed for ATM networks, when using the AnNet algorithm researchers presented AntNet+SELA [36]. The work focused on providing QoS in ATM networks for generic variable bit rate (VBR) traffic. Here virtual circuits can be established such that physical reservation of resources is possible, thus providing statistically guaranteed QoS. AntNet+SELA was never been fully tested, it was a model and not a real implementation.

3.6.2 Q-Learning

Q-learning is a popular reinforcement learning algorithm for solving Markov decision process without using an environment model. Q-Routing algorithm based on

Q-learning is proposed in [95]. In this routing mechanism, each router computes paths for the packets based on the information of the neighbouring routers. At each node, a table of Q-values are stored which depicts quality of available paths. The tables are updated every time packets are sent to its neighbouring nodes. This allows to obtain global estimate of congestion using local information. Authors in [96] use Q-Routing as a base and propose a Bi-directional low-weight clustering-based Q-routing (Bi-LCQ) for NoCs. In this mechanism, the network is split into multiple clusters and each cluster maintains a Q-table. Authors discuss that this approach leads to reducing area overhead and provides a faster way to collect and use local and global congestion information. With the aim of improving Q-routing approaches further, authors in [97] present Q-RASP, a Q-learning based NoC routing framework. Q-RASP stands for Q-routing for NoCs with region-awareness and shared path experience. Here the authors propose a cost function for choosing contention-free paths in congestion-free regions and present new update mechanism to share learning experience between packet targeting different destination but occupy same paths. An extended and revised version of Q-Routing was proposed in [98] to deal with traffic variability. In situation where links have temporary congestion, "recovery rate" model was introduced. Such that the touting algorithms give chance to paths which have recovered from temporary congestion. Both Q-R and P-QR algorithms were among the many which were compared against AntNet in [99].The authors demonstrated that AntNet showed different characteristics as it used a richer repertoire of information. For example AntNet used adaptive probabilistic models for distance estimates to determine relative goodness of each local choice, a simpler model to capture instantaneous state of link queues and stochastic decision matrix for both data and ants routing. Each node is learning on the basis of a balance between long and short-term estimates. This is done to cope with the intrinsic variability of input traffic. AntNets algorithms rely on the use of stochastic policy to route data packets which result in a better distribution of the traffic load over the network when compared against deterministic routing policy.

3.7 Overview of Limitations in Existing Work

In MCS, applications of different criticality share computation resources like multi-processors and communication resources like the NoC. A challenge is how to ensure the behaviour of lower criticality components does not adversely impact the behaviour of higher criticality components. Authors in [10] composed a survey paper on mixed criticality systems. The authors highlighted a fundamental question in MCS which is how to reconcile the two conflicting needs: The need to separate for safety and the need to share for efficient resource usage. The authors state that integration of these two areas is challenging and not easily done. To illustrate further: to incorporate flexible scheduling, dynamic partitioning is required but certified systems require at least static partitioning. According to the authors in [10] there is a need to address this mismatch.

Authors in [9], discuss challenges faced by NoCs in real-time systems. Data belonging to a real-time communication have strict deadlines and missed deadlines are as faulty as lost packets. In HRTSs, deadlines are not to be missed or it would have catastrophic consequence. Whereas SRTSs are able to tolerate a degree of missed deadlines. Therefore latency and throughput need to be predictable to ensure guarantees for real-time NoCs. This is challenging as NoCs are shard by multiple applications running concurrently. This can cause congestion and hot spots especially at high input traffic . Real-time applications are also restricted by requirements like cost and energy efficiency. Real-time MPSoCs are subjected to varying workloads and application-specific demands during its runtime [9]. Such a dynamic environment justifies investigating adaptive techniques in NoCs to improve overall performance.

A motivation for adaptive schemes is to provide flexible support to a wide variety of real-time application which have different demands during runtime [9]. Characteristics of a system can be dynamically changed at runtime to meet application requirements and in parallel have efficient resource utilization. The surveyed work in [9] showed that HRT-NoC solutions need to involve design-time phase with static characterization. And that there was a lack of adaptivity or flexibility

during runtime in real-time NoCs. Integrating runtime flexibility and adaptivity can benefit real-time NoCs

In this chapter, we have presented several existing NoCs and adaptive NoC features from topology to routing level. Mixed criticality applications can benefit from such adaptive features but the challenge is how to also ensure real-time guarantees are met. These are some of the ongoing research questions the thesis is trying to address. Some of the questions being investigated include: How can one increase runtime adaptivity and flexibility in mixed criticality NoCs? How the inherent distributed nature of the network can be used to increase adaptivity? How can adaptivity be implemented in mixed critical context while ensuring requirements are met? How can reinforcement learning based routing be utilized to benefit mixed critical systems? We address these questions by proposing adaptive and runtime features at different levels of the NoC and are presented in the next chapter.

4 Adaptive and Distributed NoC Concept

In this chapter, techniques developed at different levels of the NoC with the aim of increasing adaptability and flexibility are presented. Investigations are broadly divided into two levels: Routing and Topology level. An overview of the features developed at routing and topology level is shown in Figure. 4.1.

On the routing level, a reinforcement learning based routing using the ACO meta-heuristic [36] is proposed. ACO based routing is fully adaptive, topology agnostic and uses reinforcement learning to discover better paths over time. Such an algorithm discovers multiple paths towards its destination over time and depending on the network state, selects the path with the shortest latency. The algorithm routes packets along alternate routes when there is congestion thus reducing hotspots, and improving traffic load balance. The algorithm distributes data traffic randomly across all good routes and in this manner, traffic is spread out to prevent congestion.

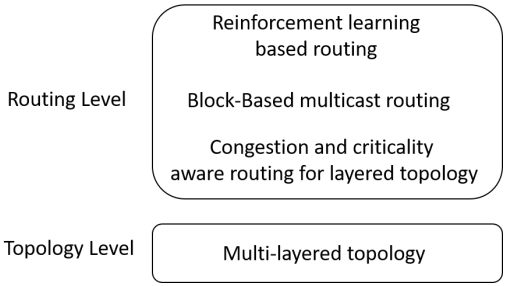


Figure 4.1: Overview of features proposed at routing and topology level

They also aid in increasing robustness to node/link failures for example by routing data around such regions. Thus aiding in better network resource utilization and improving overall throughput and latency. The potential of utilizing such a fully adaptive routing in aiding spatial isolation for mixed criticality systems is also investigated. Due to its topology agnostic feature and continuous discovery of better paths, the algorithm can discover paths at runtime within partitions and isolate traffic within them. This also aids in reducing inter-application traffic influence. Another adaptive feature introduced at the routing level, is to improve scalability of multicast routing by proposing a dynamic Block-Based multicast routing [100]. On the topology level, we propose a multi-layered NoC targeting mixed criticality systems. The aim in designing multiple layers is to decouple traffic of different levels of criticality and route them on different layers. To achieve this, runtime congestion aware adaptive routing algorithm which can route packets based on its criticality is proposed. We also investigate how the reinforcement learning based routing performs on such layered topologies.

4.1 Reinforcement Learning Based Routing

As mentioned in Section 2.8, reinforcement learning is a goal oriented learning from interaction with the environment [35]. As explained in [35], the features which make reinforcement learning different from other learning paradigms are as follows:

- The actions to be taken are unknown at the start and are discovered by trying out various actions and determining which action yields the most reward.
- Actions taken may not only effect the immediate reward but also subsequent future rewards in the long term.
- They are also closed loop problems, as the learning system's actions influences its later inputs.

Among the challenges faced in NoCs and especially NoCs targeting mixed criticality systems, we identify the potential of applying the above mentioned techniques in designing efficient, adaptive and flexible NoCs. There is potential in utilizing such features in designing adaptive and efficient routing mechanisms in mixed criticality NoCs.

4.1.1 AntNet Model

Here, we aim to utilize aspects of the AntNet routing algorithm which is based on ACO and apply to mixed criticality NoC routing. The focus is more on the reinforcement learning aspect used in the algorithm. We identify that the fully adaptive, topology agnostic ACO based routing which can discover better paths over time has potential when designing flexible mixed critical NoC systems. We start by providing an overview of the ACO based routing algorithm implemented in our work which is based on the AntNet Algorithm [36]. The following is based on the improved version AntNet-FA as described in [101]. Preliminary investigations of implementing the AntNet based routing to NoCs was conducted as part of the supervised student's work [37] and an overview is provided in the following sub-sections and in Section 4.2. Evaluations and results to present the potential of the routing algorithm to aid in traffic isolation was published in [21].

4.1.2 Data structures

Consider a network comprising of N nodes and data structures are maintained at each node k. Pheromones are organized in a matrix $\mathcal{T}^k$ as shown in Figure 4.2. When moving towards a particular destination d, the attractiveness of choosing a neighbour n is represented by a pheromone value τ_{nd}. These pheromone values are treated as measures of probability and are normalized for each destination:

$$\sum_{n \in \mathcal{N}_k} \tau_{nd} = 1 \tag{4.1}$$

		Destination		
	$\mathcal{T}$	D_1	D_2	...
Node	N_1	τ_{11}	τ_{12}	...
	N_2	τ_{21}	τ_{22}	...
	⋮	⋮	⋮	⋱

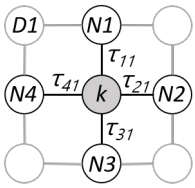


Figure 4.2: Pheromone table on the right. On the left is an example of neighbour nodes and probabilities for a destination $D1$ for a node k

where $\mathcal{N}_k$ is the set of neighbours of node k.

This pheromone matrix is supplemented by a traffic model $\mathcal{M}^k$ which represents a local view of the global traffic. Statistical parameters like mean μ, variance σ^2 and shortest travel time W over the last w observations are stored here.Additionally the algorithm also uses information of buffer levels at the output ports. To represent this, q_n describes the amount of bits waiting for transmission towards n, from which the heuristic quantities quantities l and t are derived. The quantity l is shown below and represents the attractiveness of less frequently used links.

$$l_n = 1 - \frac{q_n}{\sum_{i \in \mathcal{N}_k} q_i} \tag{4.2}$$

Quantity t estimates time it takes a data packet to travel across a connecting link (k to n) and is defined below, where bandwidth is represented by b_n and propagation delay by δ_n.

$$t_{k \to n} = \delta_n + \frac{q_n}{b_n} \tag{4.3}$$

4.1.3 Routing Ant and Data Packets

In the context of AntNet routing schemes, ants are considered as special packets that travel together with data packets but are routed according to a different policy. An ant packet starts as forward ant at a source node s and is sent to a destination node d. When the ant reaches this destination, it converts to a backward ant and

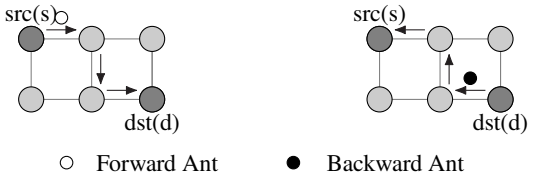


Figure 4.3: Forward and backward ant for $s = (0,0)$ and $d = (2,1)$

retraces the path back to s. This is illustrated using a simple example in Figure 4.3.

Forward ants are periodically generated at every node and the respective destinations are chosen randomly with more weight on paths which has more traffic. Forward ants store the address of each visited node in a private memory V. During routing, a neighbour $n \in \mathcal{N}_k$ is selected randomly with the probability p_{nd}, which is a combination of the corresponding pheromone value τ_{nd} and link attractiveness l_n. Here, in our implementation we denote this by the input buffer levels of downstream routers and this information is used as short-term view on the local traffic.

$$p_{nd} = \begin{cases} \dfrac{\tau_{nd} + \alpha l_n}{1 + \alpha\,(|\mathcal{N}_k| - 1)} & n \in \mathcal{N}_k \setminus V_{s \to k} \\ 0 & \text{else.} \end{cases} \tag{4.4}$$

The constant α determines the trade-off between exploration of new paths and exploitation of the ones already discovered and according to [36] is $\alpha \in [0,1]$. Neighbouring nodes which were visited already by ant packets are excluded in future exploration to avoid the formation of loops in the ant's path. If there are no options for neighbour nodes left, a random neighbour is chosen and the resulting loop is removed from V.

When the forward ant arrives at the destination, it is converted to a backward ant and using the path information in V retraces the path back to the source node. Estimated time $t_{k \to f}$ it would take to cross a link (node f to k) in the forward direction is stored in the ant's memory. On the way back, the pheromone and

traffic structures are updated as described in the next section.The backward ant is discarded/dropped once it has returned to its source node,

Data packets are routed according to probabilities based on the pheromone table. The probability of choosing the neighbour n when the destination is d is given below. The constant $\epsilon > 1$ represents how much emphasis is applied on the best paths.

$$\begin{aligned} r_{nd} &= \tau_{nd}^{\epsilon}, \\ p_{nd} &= \frac{r_{nd}}{\sum_{i \in \mathcal{N}_k} r_{id}}. \end{aligned} \tag{4.5}$$

4.1.4 Learning from Backward Ants

The pheromone table $\mathcal{T}^k$ and traffic model $\mathcal{M}^k$ are updated with the information collected by a backward ant visiting node k. The backward ant packet stores within itself time estimations for each link travelled on its forward path. These are accumulated to get an estimate T. This represents the time a data packet would take when travelling from nodes k to d:

$$T = t_{k \to d} = \sum_{i \in V_{(k+1) \to d}} t_{(i-1) \to i}. \tag{4.6}$$

Next, $\mathcal{M}^k$ can be updated according to an exponential window:

$$\begin{aligned} \mu_d' &= \mu_d + \eta(T - \mu_d) \\ \sigma'^2_d &= \sigma_d^2 + \eta\left((T - \mu_d)^2 - \sigma_d^2\right). \end{aligned} \tag{4.7}$$

The constant η represents decaying influence of past observations. This can get negligible after approximately $5/\eta$ new observations. The best recent travel time

W is updated in the current window and it is reset to the mean at the start of the next one:

$$w' = w + 1 \mod \hat{w}$$
$$W' = \begin{cases} \min(W, T) & w > 0, \\ \mu'_d & w = 0. \end{cases} \tag{4.8}$$

This results in a non-sliding window of length $\hat{w}$, that is chosen as

$$\hat{w} = c \cdot (5/\eta), \quad c \in (0, 1]. \tag{4.9}$$

The pheromone value, which corresponds to the forward ant's routing decision is increased to reward the newly discovered path. In the AntNet algorithm, the increment is based on an adaptive reinforcement factor r which reflects the quality of the path. The new pheromone value is given by

$$\tau'_{fd} = \tau_{fd} + r \cdot (1 - \tau_{fd}). \tag{4.10}$$

To evaluate the discovered path's quality, a confidence interval $I = (I_{\text{inf}}, I_{\text{sup}})$ is defined below with the confidence coefficient γ.

$$I_{\text{inf}} = W, \quad I_{\text{sup}} = \mu + \frac{\sigma}{\sqrt{(1 + w)\mathring{u}(1 - \gamma)}}, \tag{4.11}$$

The precursor of the reinforcement factor is then obtained from the relation of T as shown below.

$$\tilde{r} = c_1 \frac{I_{\text{inf}}}{T} + c_2 \frac{I_{\text{sup}} - I_{\text{inf}}}{(I_{\text{sup}} - I_{\text{inf}}) + (T - I_{\text{inf}})} \tag{4.12}$$

The ratio between the recent best time estimate and the current observation is shown by the first summand.The second one represents a correction rewarding paths that are good when compared to past observations. The reinforcement factor

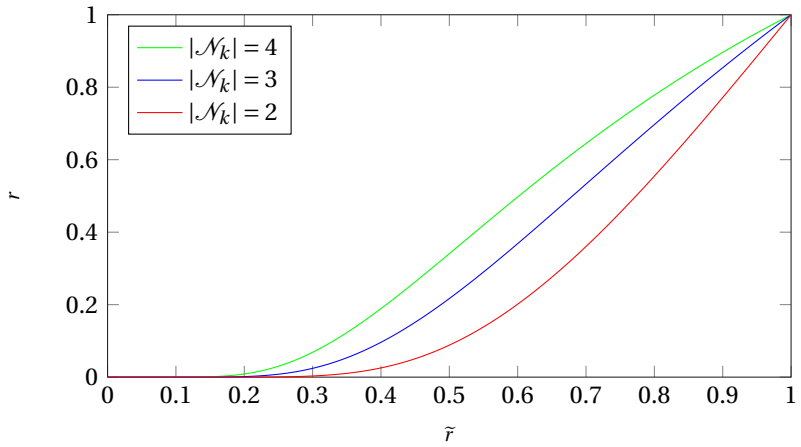


Figure 4.4: Squash function for different number of neighbours and $a = 5$ [37]

gets squashed by the squash function for emphasis of good results as shown below by the two equations.

$$r = s_n(\tilde{r}) \tag{4.13}$$

$$s_n(r) = \frac{1 + e^{a/n}}{1 + e^{a/(r \cdot n)}} \tag{4.14}$$

Parameter $n = |\mathcal{N}_k|$ adjusts the behaviour of the squash function to obtain results independent of the number of neighbours and is illustrated in Figure 4.4.

4.2 Adapting the AntNet for NoC

The AntNet algorithm was developed for telecommunication networks and is complex. To achieve an effective NoC implementation, several modifications are necessary. The previous section just highlighted certain aspects of the algorithm chosen as a basis for our NoC routing mechanism. In our implementation, we choose certain basic features of AntNet, which uses reinforcement learning

Table 4.1: Overview of AntNet's implementation parameters [37]

Parameter	Useful Domain	AntNet	Notes
α	$[0.2, 0.5]$	0.2	exploration vs. exploitation
ϵ	> 1	1.4	data exponent
η	> 0		past influence: $5/\eta$
$\hat{w}$	$c \cdot 5/\eta$	$c = 0.3$	window length
γ	$[0.6, 0.8]$	0.65	confidence level
c_1	$1 - c_2$	0.7	reinforcement factor
c_2	$[0.15, 0.35]$	0.3	weights
a	> 0	5	squash factor

mechanism. Due to the complexity of the algorithm, several modifications and simplifications have been made to make it hardware friendly. The aim is to incorporate only certain aspects of the AntNet routing mechanism, especially focusing on the reinforcement learning aspect, to develop flexible routing algorithms targeting MCS.

Pheromone quantization was examined using simulations by authors in [91]. and the results suggested that 4 bit are sufficient and higher precision resulted in diminishing returns. To decrease computational complexity, the pheromone bit size here is also set to 4 bits. A number of parameters exists in the AntNet, which influence the behaviour of the algorithm. The authors of AntNet provide a useful domain for these parameters, as well as the values used for their experiments [101] and is shown in Table 4.1. The values used in our design are obtained from these suggestions by rounding them to the nearest power of two and are shown in Table 6.3. This aids in developing a cheap hardware implementation with bit shifts.

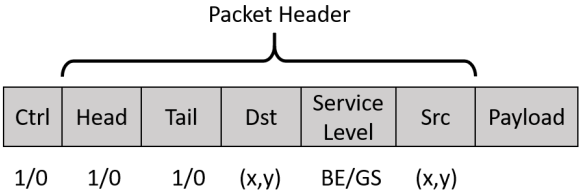


Figure 4.5: Packet format of a Head or Tail flit

Packet Header					ant	bw	pointer
dir_0	$addr_0$	dir_1	$addr_1$	dir_2	···		

Figure 4.6: Forward ant packet structure[37]

4.2.1 Ant Packets

The base NoC router utilized here is described in Section 2.3.2 [20]. Packets comprise of a header, one or more body flits and a tail flit.The packet format of a header or tail flit is illustrated in Figure 4.5. If a packet is of a single flit size, then header, data and tail information is fit inside a single flit.

In our implementation, ant packets are packets of a single fit size and the structure of forward and backward ant packets are shown in Figures 4.6 and 4.7 respectively. The header flag `ant` distinguishes ants from regular data packets and flag `bw` distinguishes forwards from backwards ants. Forward ant packet structure (figure 4.6) comprises of an array of the routing directions taken which is the private memory V, as path information needs to be stored. It is sufficient to just store either direction or address of node as they can be derived from each other. While the addresses are required during the forward ant routing to avoid loops in the paths, the directions are used for backward routing. To avoid any conversion step, both information can also be stored.

Forward ant packets are converted to backward ant packets at the destination node and inherit the memory V. Estimated link traversal time has to be stored for each hop on the way back to the source. The node addresses stored during the forward run are not required for backward routing and are therefore dropped to

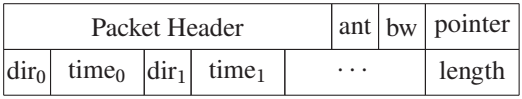


Figure 4.7: Backward ant packet structure[37]

make room for the time field in the array. This has an advantage, that the forward and backward ant packet structures (as shown in Figure 4.7) are of approximately equal length. When routing a forward ant packet, a list of all possible directions is considered. The probabilities of the ports are calculated based equation 4.4 and a direction is selected randomly. The routing decision along with the address is saved in the ant's memory. Then, the selected output link is crossed and the process repeats until the destination node is reached. When considering possible directions, it implies directions with a physical link to a neighbouring node that has not been visited yet. If there are no directions left, AntNet would choose a random direction in this case. A dedicated circuit, which detects and removes the resulting loop in the ant's memory, would be needed just for this case. To save resources, in our implementation when there are no directions left, the ant packet is dropped.

The routing algorithm routes the forward ant packets and data packets by selecting ports randomly with probabilities derived from the pheromone table. The random numbers are generated using a pseudo random number generation. Due to the pipelined design of the base router architecture, the routing decision is required to be made in a single cycle. Due to this restriction, to reduce computation overhead, the division is implemented using combinatorial logic only. A squash function is used to change the reinforcement factor. As it is challenging for this function to be translated directly to fast hardware due to the exponentials and divisions, a linear approximation is incorporated to replace the original definition since it is quantized to 4 bits.

4.3 Characteristics of ACO based Routing for NoC

Certain characteristics of the ACO based routing which can be beneficial for routing in NoCs is highlighted here.

- Decentralised Control: Actions are controlled in a de-central manner, through direct and indirect communication
- Network is explored in a random manner
- Due to the presence of a feedback loop, paths which have the shortest travel time is used more often.
- Apart from already explored paths between source and destination, alternate paths, which can be sub-optimal are also explored creating additional paths.
- Pheromone tables reflects long-term experiences of past ant packets exploring the network.
- Additionally local observation by ant packets exists to take into account changes in network state.
- Routing is topology agnostic, supports minimal and non-minimal paths.

4.3.1 Hotspot Awareness and Reliability

How can reinforcement learning concepts be applied to NoCs to improve flexibility and adaptivity? What are the challenges the algorithm can potentially address? Consider a simple example illustrated in Figure 4.8 comprising of a 4x4 NoC where communication is planned between a pair of nodes as shown. When using a static routing like XY which is minimal, the path the packets would take is shown in the first Figure. Here the minimal hop count is 2. When traffic increases in the nodes along the path, hot spots can occur which has a detrimental effect on performance. Is there a way for a routing mechanism to discover better paths

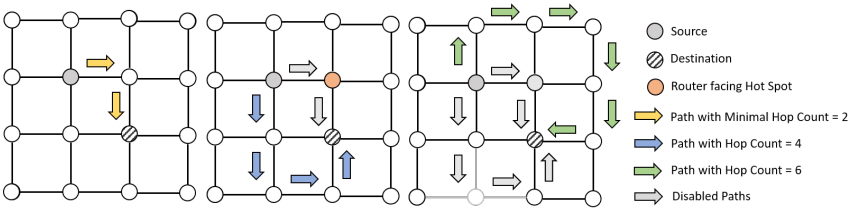


Figure 4.8: Example of paths with different hop counts between a pair of nodes in a 4x4 mesh topology

around hot spots at runtime? An example for another possible path for a given source destination pair is shown in Figure 4.8. Not just hot spots can motivate the need to discover better paths, if a router or link fails, it is beneficial if the routing algorithm can compute alternate paths at runtime. In the third Figure, the routing mechanism learns that the network state is changed (a router has failed in this example), and disables paths and discovers another feasible path.

To accomplish these, the network requires an adaptive routing scheme which discover paths between source and destination nodes at runtime depending on the network state. Is there a way for a routing mechanism to read the status of its current network state and discover new paths at runtime accordingly ? Certain situations like failure of a node or development of hot spots can be challenging to predict at design time. It would be beneficial if the routing mechanism be able to read the network state, take action to discover better shorter latency paths at runtime. During discovery, certain paths can be better than others, and over time, shorter latency paths need to be rewarded to improve overall performance. Considering these challenges, the features of reinforcement learning fits well to meet the challenges like hotspot awareness and improves reliability in cases of node failure as illustrated here.

4.3.2 Topology Agnostic Feature

Next, investigations are conducted into exploiting the topology agnostic feature of the algorithm. This is a useful feature especially if the aim is to make NoCs more flexible and adaptable to changing environments at runtime. Spatial isolation is

commonly used to reduce inter-application influence in mixed criticality systems. A simple illustration of three applications occupying a 4x4 Mesh NoC is shown in Figure 4.9a. Initially a non-critical application is occupying 7 nodes. The nodes can utilize all links in the 4x4 region to communicate with each other. Over time, two critical applications application arrive. With each arrival, the partition allocated to the non-critical application is changing to avoid influence on the critical applications. From the NoC perspective, the region allocated to non-critical applications can vary over time since priority is given to critical applications. If the NoC can adapt to the changing partition shape, compute new routes within this varying region at runtime, spatial isolation can be achieved. The challenges the NoC faces in such a scenario can include ensuring that all possible partition shapes are routable. This can result in the need to discover minimal and non-minimal routes in regular and irregular topologies. If an algorithm always routes packets using minimal paths to the destination, it can result in certain irregular topologies not supported. If using static routing, apart from reducing the runtime flexibility of the routing algorithm, it may not be able to support topologies which are not predictable at design time due to possible router failure. An example of change in topology due to router and link failure is shown in Figure 4.9b.

So is there a way to design a routing algorithm which is topology agnostic, can discover routes within unknown partitions at runtime and dynamically discover better paths over time to improve overall performance? The reinforcement learning enabled routing algorithm based on ACO is a possible way to achieve this.

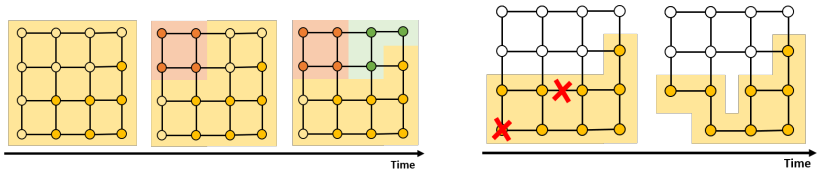


Figure 4.9: (a) An example illustrating spatial isolation overtime on a 4x4 NoC (b) Irregular topology due to router and link failures

4.3.3 Potential for Spatial Isolation

This section discusses the potential of utilizing such a routing algorithm to aid in runtime spatial isolation of resources at runtime. The network is continuously explored with probing packets and the collected information is distributed to the routing tables. The routers make decisions based on a combination of long and short-term information about the network state. This leads to a self-learning NoC, which is topology agnostic, and optimizes for throughput and latency. Since the proposed solution is at the hardware level, there is no software overhead compared to traditional table based solutions. The algorithm does not require knowledge of the shapes at design time and can discover non-minimal paths. It has a full coverage of shapes and computes minimal and non-minimal routes.

As the algorithm is topology agnostic, it is suitable for scenarios where spatial isolation is implemented for MCS with varying partition regions. In the ACO based routing, the network is explored with special packets and the application data packets are routed based on the information obtained. This separation allows for exploration of paths that are considered futile but may have become viable because of a change in traffic. This results in the exploration of seemingly sub-optimal (non-minimal) paths as well. Thus the algorithm also supports minimal and non-minimal paths within regular and irregular topologies.

4.3.4 Traffic Isolation

How can we use this reinforcement learning based topology agnostic routing to aid in traffic isolation? When a certain region is occupied by a critical application the border routers are informed to stop accepting any packets from outside this region. For example, if an exploring ant packet attempts to enter the region it is dropped. This results in the algorithm learning that the path is not feasible and attempts another path towards its destination. Forward ants, when discovering paths, randomly choose a direction at each node to reach the destination. Once it reaches the destination, the timing and path information, which is collected is

transferred to the returning backward ant, which distributes this information along the path. If an ant fails to find a path, there is no backward ant being generated and thus information of this failed path is not updated in the pheromone tables. Which implies that the pheromone value in the direction of the failed path is not increased. Data packets are then routed according to the probabilities derived from the pheromone table.

The ACO based routing can be utilized by non-critical applications or applications which do not require guarantees. The critical applications can utilize GS communication and deterministic routing algorithms like the XY to ensure guarantees. For critical applications, usually the shape of the region they consume is known at design time. When they start running on the system, edge of the region routers are communicated so a boundary is established. Let us illustrate with a simple example where an ant packet is traveling from $(0,0)$ to $(3,3)$ in a 4x4 design as shown in Figure 4.10. Initially no other traffic is present. In the first case, all links in the 4x4 region are allowed to be used to reach the target. An ant packet was injected at node (0,0) periodically into the network. Over time, paths were discovered and are highlighted in red. Most ant packets chose a path with minimal hop count. The number adjacent to the link indicate number of ants travelled on it. The algorithm over time optimizes for latency due to the traffic model integrated in the algorithm. It reads short time information like local buffer information and global information like travel times of ants.

In the second case, an application has arrived and is running on four nodes in the centre of the network and does not allow packets entering the region. The ant packets attempt to enter and are dropped and discover new paths to the destination as shown. Two paths from (0,0) circumventing the 2x2 region are discovered and since both had the same number of hop count, one of them was chosen randomly.

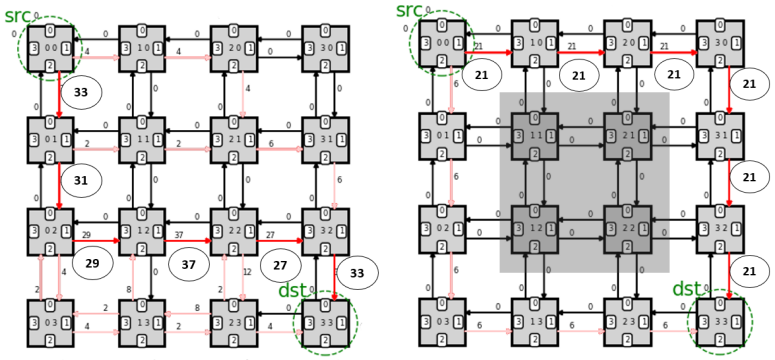


Figure 4.10: Ant packets discovering paths from (0,0) to (3,3) [21]

4.4 Runtime Adaptive and Scalable Multicast Routing

In this section, we describe the features proposed to improve adaptivity and scalability of multicast routing schemes. Most NoCs focus on unicast traffic, where communication is between a single source and a single destination. Multicast traffic where communication is between one source and multiple destinations is extensively used in large-scale multiprocessor systems. Authors in [102] show that multicast based mesh networks provide high performance/cost ratio and are most suitable when implementing configurable neural networks. Other examples are applications using data-parallel programming model and Single-Program Multiple-Data (SPMD) programming model in which the same program is run on multiple processors in parallel. Another motivation for a multicast feature is to support coherency protocols for distributed shared-memory paradigm [70] [74] [103]. Usage of multiple unicast packets in the above examples degrades the performance and is an inefficient use of resources. These applications can greatly benefit from efficient hardware multicast features in NoCs. Communication in such systems is usually done in software using Message Passing Interface (MPI). Significant overhead can be decreased using hardware techniques. Though few hardware multicast techniques in NoCs exist [66], a NoC based multicast solution

which is scalable, flexible and dynamic has not been shown yet. We focus on these features and present a block-based multicast solution for mesh networks.

In this section, we discuss a multicast technique which is scalable, with lower packet overhead and latency. This is an important feature considering that the number of nodes in SoCs and complexity of applications using such systems is continuously increasing. It is a hardware block-based multicast routing for mesh NoCs which incorporate dynamic bit vector index mapping onto nodes. A block of potential destination nodes is defined at runtime. Dimensions of the block and the bit vector are stored in the packets. This technique is advantageous when destinations are located close together. It provides runtime support for both unicast and multicast routing with few changes and provides flexibility in the platform.

4.4.1 Block-Based Multicast Routing Concept

When a source nodes needs to send the same data to a group of destination nodes, a block is identified which contains all the destination nodes. The block is defined by two node addresses present diagonally opposite to each other. This is illustrated in Fig. 4.11 using an example of a 4x4 Mesh. A source node needs to send a message to 6 nodes highlighted by a rectangle region which we refer to as a block. This block is defined by the source by using a Start Destination Node (DstS) and a End Destination Node (DstE). A packet is built at the source and forwarded by the routers according to Dimension-ordered (XY) routing to DstS. At DstS the packet is duplicated and three possible directions are identified: Local (L), Branch (B) and Tree (T). The Local direction forwards a packet to the local port and the Tree direction is part of the path to DstE. The Branch direction branches away from this Tree path. So at DstS, the packet is duplicated thrice and forwarded in each of the L, B and T directions. Duplication occurs at each node in the block and the number of duplications depends on the position of the node. Paths taken by the packet within and outside the block is illustrated in the Fig. 4.11.

The Block-Based multicast packet header contains information of DstE and DstS as shown in Figure. 4.12b. For comparison, a multicast packet header which uses

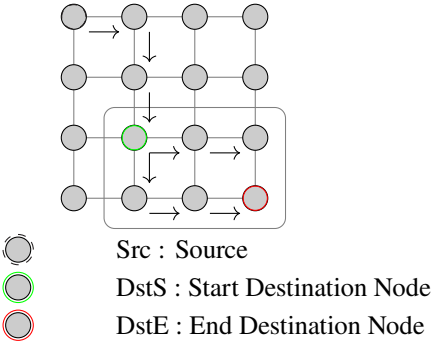
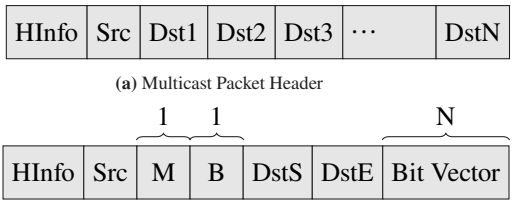


Figure 4.11: A 4x4 Mesh with 6 destinations in a block defined by DstS and DstE [100]



(a) Multicast Packet Header

(b) Block Multicast Packet Header

Figure 4.12: Comparison of all destination encoding in multicast packet header and block-based multicast packet header for N destinations [100]

all destination encoding, where all destination address are present is shown in Figure. 4.12a. A Multicast bit (M) indicates if it is a unicast or a multicast packet. A 'bit vector' bit (B) informs the router if a bit vector field is present or not and this field is followed by DstS, DstE and the Bit Vector (BV) field. If the B field is set then it implies that not every node in the block is a destination and a BV field is present to indicate which are the destination nodes. Number of bits in the BV field is the number of nodes inside the block defined by DstS and DstE. Since the block can be defined at runtime, the length of BV is dynamic.

This is explained using an example shown in Figure. 4.13. Addresses of nodes are its X and Y co-ordinates, source node is (0,0) and destinations are (1,2),(2,2),(1,3) and (3,3). A block is defined by DstS (1,2) and DstE (3,3) and there are 4

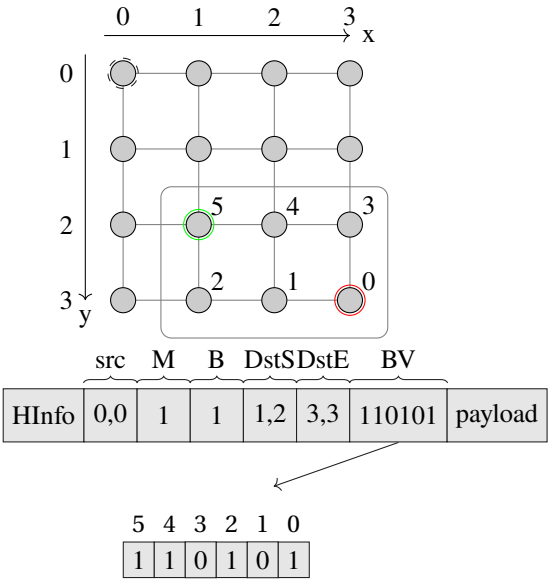


Figure 4.13: A 4x4 Mesh with 4 destination nodes (0,2,4,5) in a block defined by DstS, DstE and a BV [100]

destinations within this block identified by the BV. BV indices are mapped to the nodes within the block as shown in Figure. 4.13.

The router at each node processes DstS, DstE and BV information in the packet header and determines if a node is outside the block or inside the block. If the current node is outside the block, DoR routing algorithm is chosen. This occurs until packet reaches DstS. If a node is found to be inside the block, BV is processed and the relevant BV index of the node is computed and read. Using this information the packet is duplicated accordingly. For example at DstS, BV index is computed to be 5 and its value is '1'. The packet is then duplicated thrice (one each for B,L,T). At node 3, the BV value is 0 so there is no duplication and the packet is terminated. At node 1, the BV value is 0, so the packet is only forwarded in T direction towards DstE. At node 2, the BV value is 1, so there are two duplications done for L and T directions.

In generic terms, the total number of nodes in a block is represented by 'N'. A source sends to 'i' destinations where $1 \leq i \leq N$. When $i = N$, all nodes in the block are destinations and it then becomes a region based broadcast. When $i < N$ only some nodes in the block are destinations and a BV field of length N is used to identify which of the nodes in the block are destinations. BV index (BV_i) is computed using addresses of DstS (X_s,Y_s), DstE (X_e,Y_e) and current node (X_c,Y_c) as shown in Eq (4.15).

$$BV_i = \mid X_c - X_e \mid + \mid Y_c - Y_e \mid * (\mid X_s - X_e \mid + 1) \tag{4.15}$$

Address Overhead (AO) for a 2D mesh is shown in Equation (4.16) for one block where all nodes in it are destination nodes. One bit each for M and B fields and x_b and y_b are bits required for X and Y co-ordinates respectively. The AO in case of using the BV field can be calculated according to Eq (4.17).

$$AO = 2 + 2 * (x_b + y_b) \tag{4.16}$$

$$AO = 2 + 2 * (x_b + y_b) + (\mid X_s - X_e \mid + 1) * (\mid Y_s - Y_e \mid + 1) \tag{4.17}$$

An illustration of the BV index to node mapping when the orientation of source and destination regions change is shown in Fig. 4.14. Source node is at the center and quadrants are used to illustrate the mapping better. Nearest destination node to the source node is selected to be DstS (in green) while the farthest as DstE (in red). The BV size is 9 and paths of packets taken within the block is shown as well.

Since regular packets using DoR are also travelling in the NoC along with the multicast packets, deadlocks are a problem. This is because turns that are usually not allowed are possible in the paths of multicast packets within the block. To avoid this, multicast packets and regular packets use different virtual networks. The multicast packets travel in a different virtual channel from regular packets. Thus the turns are constrained in a different virtual network and loops are avoided. Each packet is associated with a virtual channel ID and the router takes this into

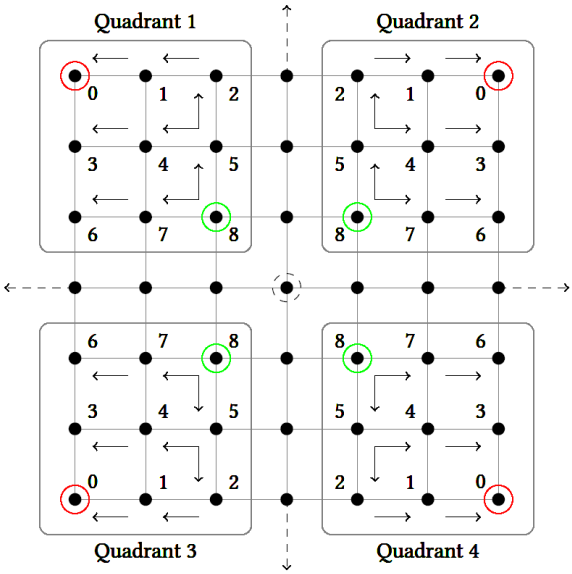


Figure 4.14: An example illustrating the mapping of BV index onto the nodes using quadrants for a 7x7 network and 9 destinations [100]

consideration when choosing the output virtual channel at each output port. The Block-Based multicast routing work was published in [100].

4.5 Multi-Layered NoCs with Congestion Aware Adaptive Routing

In this section, we take a look at the topology level improvements proposed targeting mixed criticality systems. Mesh topology is commonly implemented in a NoC due to its scalability, ease of implementation and uniform nature. A static routing is usually preferred in the NoC for critical applications as they require guarantees at design time. Therefore it is challenging to utilize adaptive routing in mixed criticality NoCs. In this section, we present design and implementation of a multi-layered NoC with an adaptive routing algorithm designed to be beneficial

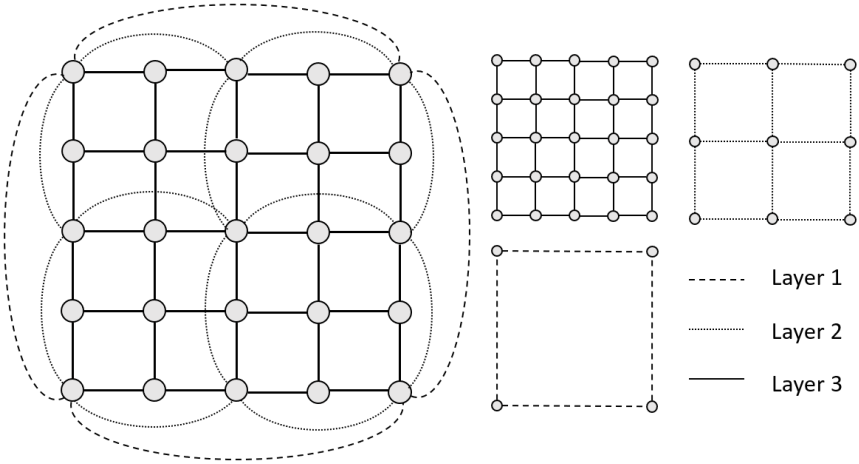


Figure 4.15: Multi-layered topology under test [106]

for mixed criticality systems. The congestion-aware, adaptive routing algorithms was implemented and evaluated as part of a supervised students work [104] and published in [105]. An overview is presented in this section.

The proposed topology is mesh based, multi-layered and hierarchical. An illustration of the multi-layered topology is shown in Figure 4.15. The motivation for mesh based topology is to maintain uniformity and ease of implementation. The routing algorithm is adaptive, monitors the environment to identify paths with less traffic to avoid congestion and routes packets accordingly based on the criticality of the packets. The algorithm supports applications which require guarantees by forwarding them on known paths. For packets which do not require guarantees, paths are chosen depending on the availability. The concepts in this section are summarized below:

- Design of a multi-layered Hierarchical NoC targeting mixed criticality systems.

- A runtime congestion aware adaptive routing algorithm which routes packets on different paths depending on its criticality on such a multi-layered NoC to improve latency and throughput.

In MCS, critical and non-critical packets can share the same communication resources like links and routers. It can be beneficial, if the paths of critical and non-critical packets are decoupled. One option is to have dedicated links using virtual channels. In such cases they still share router resources and the arbitration policy on such routers can still have an impact on critical packets due to non-critical packets. Another option is to have dedicated links and routers for the critical packets. That requires a high amount of resources and can require the knowledge of mapping at design time. We aim to combine these approaches by implementing layers in a NoC and having parts of the layers occupied exclusively by critical applications. We also take into account that topologies need to be generic and uniform such that it can be used by a large set of applications and are not customized to a small set of applications.

By using multiple network layers, using an adaptive routing algorithm, we utilize the link and router resources to direct critical and non-critical paths along different paths to reduce influence between critical and non-critical applications. We aim to utilize the base mesh layer for non-critical packets and utilize upper layers with lower hop counts for more critical packets. Some of the distinguishing characteristics are summarized here

- Critical applications implemented on several nodes and which spread across the SoC have higher priority and are assigned paths of lesser hop counts across layers to communicate with each other. Non-critical packets share resources when available or are directed on paths with longer hop counts.
- A non table based adaptive routing algorithm is proposed for a multi-layered NoCs which is scalable and flexible.
- Memory and IO Tiles which are generally at fixed locations and are sometimes spread across the SoC can benefit from such a NoC by having heavy

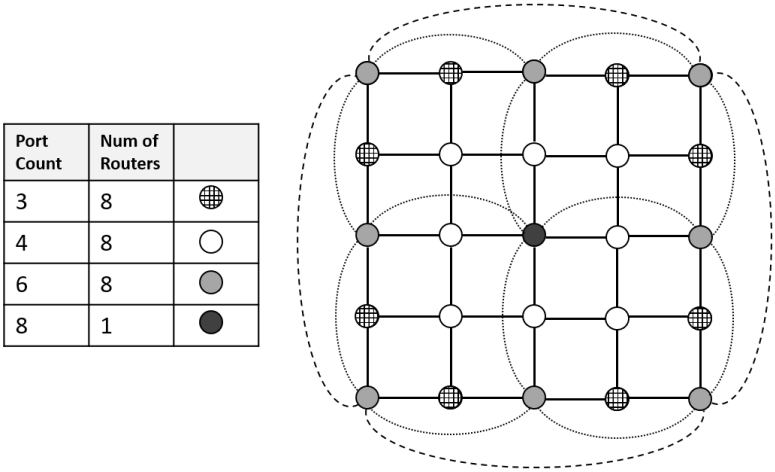


Figure 4.16: Variation in router count and potential of path diversity in a 5x5, three layer network [106]

traffic to and from such nodes be restricted to certain paths to reduce congestion.

Such a multi-layered NoC comes with the cost of increase in router resources, with the increase in links in routers which connect the different layers. So a trade-off between the resource consumption and benefit in performance is desired.

4.5.1 Multi-Layered Topology Concept

The topology we are using for our approach is a multi-layered mesh as shown in Figure 4.16. The basis for communication is a base mesh layer connecting all of the nodes. Alternating nodes on this layer are additionally connected using another network with a mesh topology to create a second layer. And alternating nodes on the second layer are connected to create the third and so on.

To demonstrate the advantages of such a topology and the supporting routing algorithm we consider an example of a three layers, 5x5 variant as illustrated in

the Figure 4.17. The network interconnects 25 tiles. Meanwhile the number of ports of a router varies depending on its position in the network. The minimum number is 4 (example at Tile 1, three in E, S, W direction and a local port) and the maximum possible number of ports is 9 (example at Tile 12, 2 in each of the cardinal direction, plus 1 local port).

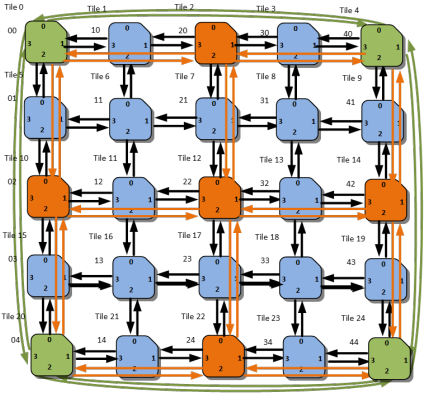


Figure 4.17: A 5x5 3 layered NoC [105]

In the proposed NoC, priorities are assigned to the layers as shown below. For dimensions of 5x5 and 3 layers, the priority of layers is provided below.

$$Layer_1 > Layer_2 > Layer_3 \tag{4.18}$$

For a general n-layered network, Layer_n is the base layer which connects all the nodes and Layer_1 is the top most layer.

$$Layer_1 > Layer_2 > ..... > Layer_n \tag{4.19}$$

In this example the nodes are classified into three classes according to their location.

- Class 1: Routers connected by the highest priority layer (here it is the 2x2 mesh). Highlighted in green.
- Class 2: Routers connected using the intermediate layer (here it is the 3x3 mesh) which do not belong to Class 1. Highlighted in orange.
- Class 3: Routers connected using the base layer (here it is the 5x5 mesh), which do not belong to Class 1 and Class 2.

4.5.2 Adaptive Congestion Aware Routing Concept

We start by explaining the routing algorithm without a congestion avoidance feature. The path with shortest hop count between a source and destination nodes is selected across layers. Since the paths can be computed at design time, one way to implement such an algorithm can be by using a table based routing scheme. For this method the paths are computed at design time for a certain source and destination pair. Routing tables are implemented at each router, which store the output port to be chosen for a certain destination node. The values in the routing tables are fixed. A disadvantage of such an implementation is that tables do not scale well and are not flexible. For a 9x9 mesh, each router needs to store a table containing the output port for the 80 potential destinations. To avoid the implementation of routing tables and to have flexibility we present a routing algorithm which computes the path at run time. This makes the implementation of the algorithm lightweight, flexible and scalable.

The routing algorithm at every router forwards the packets on paths which use the least number of hops. To achieve this, a Near Node (NN) function is defined. That is carried out as follows. Consider a source (x_s, y_s) and destination pair (x_d, y_d). For a node (x, y) the NN function computes the nearest node (x_n, y_n) located between (x_s, y_s) and (x_d, y_d) which connects to the higher layers. The NN function is used to compute nearest nodes for the source and destination nodes. This results in (x_{sn}, y_{sn}) and (x_{dn}, y_{dn}) which are nearest nodes to source and destination nodes respectively and satisfy equations 4.20 and 4.21 .

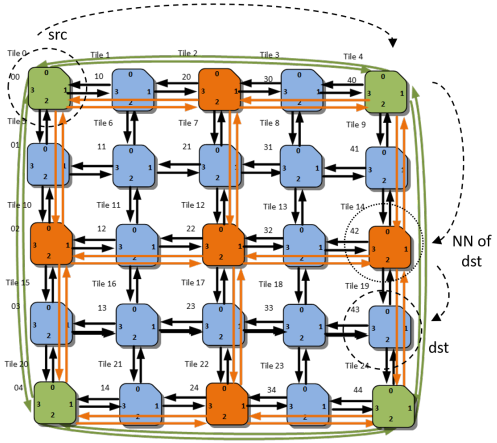


Figure 4.18: Example when source is (0,0), destination is (4,3) and Nearest Node for destination is computed to be (4,2)[105]

$$(x_s < x_{sn} < x_d) \text{ or } (x_s > x_{sn} > x_d) \text{ and } (y_s < y_{sn} < y_d) \text{ or } (y_s > y_{sn} > y_d) \quad (4.20)$$

$$(x_s < x_{dn} < x_d) \text{ or } (x_s > x_{dn} > x_d) \text{ and } (y_s < y_{dn} < y_d) \text{ or } (y_s > y_{dn} > y_d) \quad (4.21)$$

Now let us use a general example to show how a packet travels between a source (x_s, y_s) and destination node (x_d, y_d) using this NN function. For reference, let us consider the same for XY routing and only consider the base layer. A packet first travels in the X direction towards the destination and then the Y direction. It utilizes the links only in the base layer. The proposed routing algorithm now seeks to utilize the links in the higher layers which have shorter hop counts. The NN function determines the nearest node to the source (x_{sn}, y_{sn}) and nearest node to the destination node (x_{dn}, y_{dn}). The entire path between the source and destination node is now divided into three sub-paths due to these nodes. Now the packets travel as shown in equation 4.22.

$$(x_s, y_s) \rightarrow (x_{sn}, y_{sn}) \rightarrow (x_{dn}, y_{dn}) \rightarrow (x_d, y_d) \tag{4.22}$$

Within each of the sub-paths, the XY routing algorithm is utilized to travel between nodes. For example, between (x_s, y_s) and (x_{sn}, y_{sn}), multiple paths are possible, but to achieve minimal routing between these nodes and to keep the algorithm simple, XY routing is used. Similarly between (x_{sn}, y_{sn}) and (x_{dn}, y_{dn}), XY routing is used to transport the packets.

Let us explain this by using an example. A packet is transported from $(0,0)$ to $(4,3)$. The destination node is $(4,3)$, its possible nearby nodes are $(4,2)$ and $(4,4)$. This is illustrated in Figure 4.18. The NN function always selects a node, which is located between the source and destination node. In this example, the near node of $(4,3)$ is $(4,2)$. It is located on a higher layer and is located between source and destination node, and thus will be chosen as near node of the destination node. Since the source node already connects the highest layer, the nearest node of the source node is the source node itself. Now the packet travels using the path $(0,0) \rightarrow (4,0) \rightarrow (4,2) \rightarrow (4,3)$. Between $(0,0)$ and $(4,0)$ a layer 1 link is used. Between $(4,0)$ and $(4,2)$ a layer 2 link is used. And between $(4,2)$ and $(4,3)$ the layer 3 link is used.

In general, to summarize, the routing algorithm prioritizes higher layers to forward the packet. The NN function is used to determine intermediate nodes and the path is broken into sub-paths. The general rules the NN function follows are as follows for the example of the 5x5 three layer network:

- Routers of Class 1 don't use the NN function, because they are already on the Layer with highest priority.
- The nearby node that is computed by the NN function for routers of Class 2 are members of Class 1, and the nearby node for routers of Class 3 belong to Class 1 or 2.

The algorithm just described is a static algorithm, the packets are always routed on the same paths for a certain pair of source and destination nodes. The paths do not

change even if there is less or more traffic on those paths. In this algorithm, paths with low hop count are desired by packets generated by all nodes. When there is high traffic, there is a high possibility that the nodes connecting the higher layers experience more traffic since all packets are attempting to travel on the paths with the shortest hop count. The resulting delay increases with the increase in injection rate of each node due to the increase in the waiting times within the router. To solve this issue, a congestion awareness feature is integrated into the algorithm. The algorithm reads the input buffer status of the downstream routers and makes a decision among the possible output ports. In the results section we present latency and throughput results of the algorithm with and without the congestion avoidance feature to present the difference in performance. The results show improvement when the congestion avoidance feature is integrated in the algorithm. Next, the congestion avoidance feature is elaborated. Let us take an example, consider a packet from node $(0,0)$ to $(3,0)$ as shown in Figure 4.19. Here only a section of the network is shown for ease of explaining. There are multiple paths possible across the layers. The first path uses only the top layer (one hop), the second path uses the intermediate layer and base layer (2 hops) and the third path uses only the base layer (3 hops).

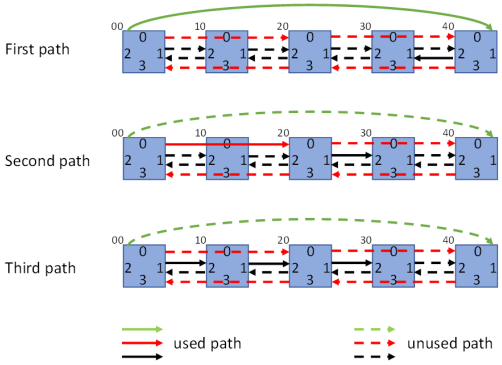


Figure 4.19: Possible paths from node 00 to 30[105]

At router $(0,0)$, three output ports are possible to reach the destination. The path with the shortest hop count is using the port connecting the top layer. The

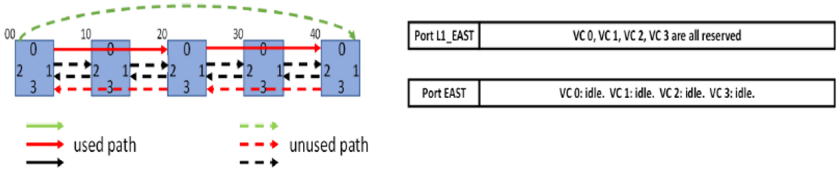


Figure 4.20: Virtual channel status [105]

next shortest path is using the port connecting layer 2 and the longest is the port connecting the base layer. In the version without congestion avoidance, the packets always chose the port connecting the top layer which results in the lowest hop count. In the second version, with congestion avoidance, the router reads the input buffer status of the downstream routers and makes a choice according to the priority of the ports. The priority of the ports are as follows:

$$port_{layer1} > port_{layer2} > port_{layer3} \tag{4.23}$$

When the input buffer in the downstream router via port $port_{layer1}$ is full then the port $port_{layer2}$ is considered. If this input buffer is full as well then port $port_{layer3}$ is considered. When this is full as well the process repeats until space is available in any of the downstream input buffers. This results in less congestion around the routers connecting higher layers and also uses the base and lower layers more efficiently. This aims to evenly distribute packets across layers. An example to demonstrate the working of the congestion aware routing is presented in Figure 4.20 when the source node is (0,0) and destination is (4,0). There are two ports possible to choose from at the source and depending on the virtual channel status in the downstream route, an output port is chosen.

In version 1 of the algorithm, critical and non-critical packets are treated the same by the routing algorithm. In version 2, critical packets are always routed on the paths with the shortest hop counts and the non-critical packets can be routed on paths with the shortest or longer hop counts. In high traffic scenarios, to avoid

influence of non-critical packets on congestion of critical packets, non critical packets are routed across higher hop count paths. A field in the header flit of the packet is reserved to identify if a packet is critical or not, and the router processes the packet and computed the output port accordingly.

4.6 Summary

In this chapter, an overview of the concepts is proposed to increase adaptivity at routing and topology level. The working of the reinforcement learning enabled routing algorithm based on ACO for NoCs was described. This included an overview of the original AntNet algorithm which targeted telecommunication networks. The algorithm was simplified, reinforcement learning concepts was focused upon more and adapted to NoCs. Characteristics of this ACO based routing algorithm was highlighted like topology agnostic feature, decentralised control and ability to adapt based on network state and discover optimal routes at runtime. The potential of exploiting such features to aid in flexible spatial isolation was discussed. Other adaptive routing mechanism concepts presented include adaptive Block-Based multicast routing. At the topology level, multi-layered NoC with supporting congestion and criticality aware adaptive routing algorithm which targeted mixed criticality traffic was discussed The aim here was to decouple traffic and transmit non-critical packets on short or longer paths across network layers based on local network congestion. In the next chapter, the formalization of the concepts of the proposed reinforcement learning based routing and how it is utilized in aiding in spatial isolation in mixed critical systems is presented.

5 Runtime Adaptive Spatial Isolation Framework

In this chapter, we present a Distributed Reinforcement learning Enabled Adaptive Mixed-Critical (DREAM) NoC and supporting framework, which uses the reinforcement learning enabled routing algorithm based on ACO introduced in the previous chapter. The DREAM framework, utilizes the adaptive and dynamic concepts of the routing algorithm to aid in runtime adaptive spatial isolation targeting mixed-critical platforms. The aim for the framework is to aid in efficient partitioning of NoC resources. We also present the formalization of the concepts of the proposed reinforcement learning based routing. We later extend the proposed reinforcement learning enabled routing to multi-layered NoCs.

5.1 Characteristics of Mixed Criticality Systems

Performance on mixed criticality systems can be improved if the NoC is flexible and support provisioning and isolation of resources [107]. An approach when implementing high criticality applications traditionally is to have dedicated resources allocated, to avoid influence of non-critical applications. Techniques like temporal and spatial partitioning of resources are commonly used in MCS to ensure real-time requirements are met [108]. Partitioning is commonly employed to reduce influence of non-critical applications over performance of critical applications. Examples include implementing time and space partitioning. In this work we investigate how to aid in flexible spatial isolation at runtime. Efficient sharing

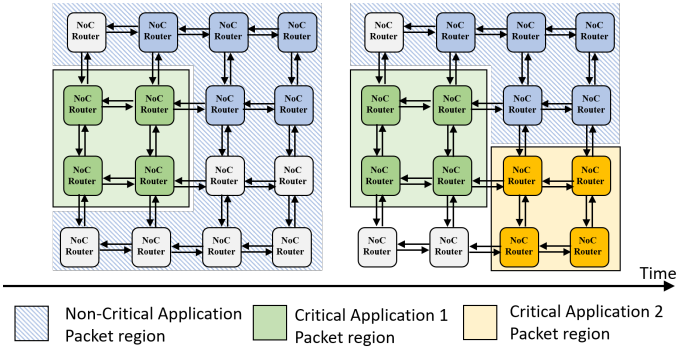


Figure 5.1: Example of multiple applications sharing NoC resources over time and maintaining spatial isolation on a 4x4 NoC [109]

and partitioning of processing and communication resources on such platforms running applications of different criticality levels can get challenging. Some of the challenges of mixed criticality NoCs include ensuring QoS requirements like throughput and latency of critical applications are met. It is desirable if the NoC can support efficient and flexible spatial partitioning [108] while ensuring performance guarantees. Traffic of varying criticality is travelling on the interconnect and here the objective of the network is to adaptively aid in isolating traffic at runtime to increase overall performance while ensuring guarantees.

A commonly used NoC topology is mesh due to its ease of implementation and regularity. When concurrent applications are sharing this interconnect, spatial partitioning can result in regions of irregular topologies. This is illustrated using an example as shown in Figure 5.1. In this example, three applications of varying criticality are sharing the network resources over time on a 4x4 mesh topology. Each router in the figure is connected to a processing tile. In this illustration, only the routers are shown for simplicity. Initially two applications are running as shown on two partitions. A critical application is occupying 4 nodes and the traffic is restricted within this region. A non-critical application is running on 5 nodes and its traffic is restricted to the region highlighted in blue. Over time another critical application arrives and the non-critical application region decreases to make space for the higher priority critical application.

From the NoC point of view, the region occupied by the non-critical application is varying over time. If the NoC can adapt to the changing partition shape, support communication within this region by determining routes within this new region at runtime, traffic isolation can be achieved. The challenges for the NoC here include discovering paths for new partitions which can be unknown at design time and supporting routing for all possible partition shapes. This provides a motivation to investigate efficient topology agnostic routing algorithms. Another motivation for topology agnostic routing is to ensure reliability in case of faults. For example, if faulty routers or wires occur, this results in a section of the network being defective while the rest of the network being fully functional. This results in irregular topologies which are not determined at design time. It is desirable that the NoC can discover new paths around the faulty sections of the network at runtime to communicate between rest of the network. To meet these challenges, we present **D**istributed **R**einforcement learning **E**nabled **A**daptive **M**ixed-Critical (DREAM) NoC and supporting framework. The network utilizes the proposed topology agnostic, reinforcement learning enabled routing algorithm based on ACO presented in Chapter 4. An aim of the routing algorithm to aid in traffic isolation within partitions and improve overall performance. The concept and evaluation of DREAM NoC and supporting framework was published in [13].

5.2 Overview of DREAM NoC and Framework

The DREAM NoC comprises of routers running the reinforcement learning enabled routing based on ACO, hereafter referred to as DREAM routing for simplicity purpose. The base router used is the *i*NoC router described in Section 2.3.2 [20] and the ACO based routing was integrated into it. The routing algorithm based on ACO [101] chooses paths based on a combination of long and short-term information about the network state resulting in a self-learning routing algorithm. It is topology agnostic and optimizes for throughput and latency. The topologies need not be known at design time and tables at each node which aid in routing are

built in a distributed manner at runtime. If better paths are discovered over time, the tables are updated accordingly.

The DREAM framework is designed to aid in flexible adaptive spatial isolation at runtime. For critical applications, the partition shapes are typically known at design time as throughput and latency guarantees need to be met. Such applications use GS communication which uses deterministic routing algorithms. For mesh based topologies it is XY routing and for others it is a table based routing where paths are deterministic. The tables for such cases are built at design time based on a topology agnostic routing like UD. Here we focus on routing non-critical packets and to aid in flexible isolation of its traffic. These applications use BE communication where ACO based routing is used and the paths are not static.

5.2.1 Scalability Analysis

There exists topology agnostic routing algorithms like UD to determine paths in irregular topologies. If all possible partition shapes are known at design time, topology agnostic routing algorithms like UD can be used to compute routing tables for each possible topology. Table based routing does not scale well with the increase in network size. Here we analyse the memory requirements when a router is part of different topologies over time and thus needs to store routing information for all possible topologies.

Consider a simple example of 4 routers interconnected using a 2x2 mesh. All possible partitions into regions are shown in Figure 5.2a. For a simple 2x2 there are 12 possibilities. Focusing on the top left router, the number of unique topologies it belongs to is 7 as illustrated in Figure 5.2b. A single router topology is also included for completeness. This results in each router storing 7 different routing tables, or routing tables need to be updated each time a topology changes. This requires that all possible routing table information be stored externally and a table is updated at each router using an external network or use a centralised router for example. The number of possible partitions increases to 1434 for a

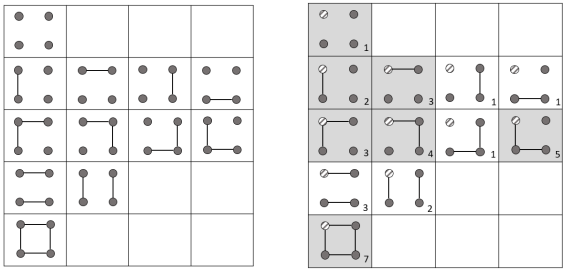


Figure 5.2: All possible orthogonal partitions in a 2x2 (a). The number of unique partitions the top left node belongs to (b) [13]

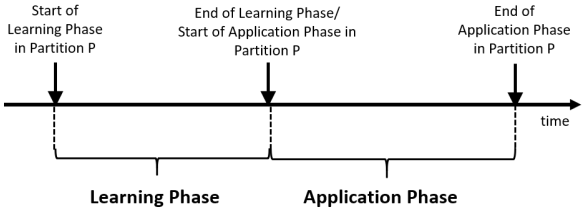


Figure 5.3: Overview of DREAM framework for a partition P

3x3 mesh topology [110]. This does not scale well and can incur high resource overhead.

If there is a topology agnostic routing like the ACO based routing which can discover paths at runtime, each router stores only one table. The trade-off is that there is a learning phase overhead where the routing tables (here we consider pheromone tables for ACO based routing as routing tables) are built before the application can transmit the packets to avoid packet loss. The algorithm due to its inherent dynamic nature also has the advantage that based on the fluctuating traffic pattern, better paths are discovered over time thus improving overall throughput and latency. The paths are not deterministic and continuously discover optimal paths. It also improves reliability when a node fails, new paths can be discovered around it.

5.2.2 Outline of DREAM Framework

An overview of the proposed DREAM framework is illustrated in Figure 5.3. It is broadly divided into Learning Phase (LP) and Application Phase (AP). During LP, special ant packets are generated at each node to discover paths between every pair of nodes within a network partition *P*. This phase is dedicated to discovering paths within a partition, application packets are not injected during this time. Special ant packets here are forward and backward ant packets which are explained in Section 4.1. The forward ants generated at each node attempts to discover a path to a particular destination and if successful returns as a backward ant to the source ode. During this return path, pheromone tables across all routers in a given partition are updated with this new path information in a distributed manner. If more backward ants per node are received, better paths are discovered over time and pheromone tables at the routers are updated accordingly. This is due to the reinforcement learning nature of the routing algorithms, over time better paths are *rewarded*. This can increase the duration of the learning phase which is an overhead. To ensure low learning phase overhead, we define learning phase such that if at least one path is discovered between every source and destination node pair, the learning phase is said to be completed.

Once the learning phase ends, this triggers the start of Application Phase (AP). Where application packets are injected into the network. In the learning phase, forward ants are injected at each node at a certain rate (for example every few cycles to ensure low overhead). Once a node discovers a path to a certain destination, it stops injecting forward ants to that destination to decrease traffic and thereby decreasing learning phase time overhead. The paths discovered at the end of this phase can be sub-optimal as the learning has just started and data traffic is not present. Better paths are discovered due to the combination of long term and local traffic information. Therefore, during the application phase, forward ants are injected at all nodes once again at specified intervals, to discover better paths over time based on the data traffic fluctuations as well.

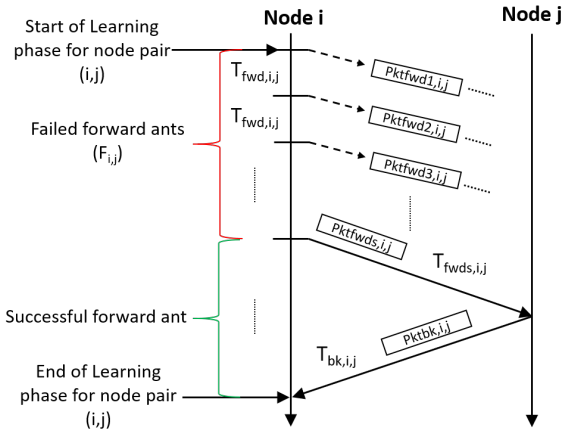


Figure 5.4: Learning time between a single pair of node, from node i to node j [106]

5.2.3 Learning Phase Analysis

Learning phase is an overhead which can vary depending on the number of nodes and the regularity of nodes within a partition.Learning Phase here for a given partition is defined as follows.

Learning Phase*The Learning Phase is defined as the time required to discover at least one path between every pair of nodes within a network partition.*

In a partition, each node injects forward ant packets at certain intervals targeting a random destination node. The time it takes to discover a path from node *i* to node *j* is analysed next. Consider Figure 5.4. Node *i* injects forward ants targeting destination node *j* until one is successful. This implies a successful path has been discovered from node *i* to *j*. In the figure, $T_{fwd,i,j}$ represents the interval between forward packets injected from node *i* to node *j*. The learning time $T_{LT,i,j}$ is the time it takes to discover one path from node *i* to node *j*. This is given by equation 5.1.

$$T_{LT,i,j} = \sum_{f=1}^{F_{ij}} T_{fwd,i,j,f} + T_{fwds,i,j} + T_{bk,i,j} \tag{5.1}$$

The time taken for the successful forward ant to travel from node i and reach the destination node j is represented by $T_{fwds,i,j}$. Once it reaches the destination it is injected right back into the network as a backward packet towards the source node. The time it takes for the backward ant to reach the source node i from node j is given by $T_{bk,i,j}$. A buffer at the destination node receives the forward packet and injects it back into the network, the time spent in the buffer here is considered part of $T_{bk,i,j}$ for simplicity. The number of failed forward ant packets from node i to j is given by F_{ij}. The time interval between their injection is represented in a generic manner by $T_{fwd,i,j,f}$.

In the implemented design, the interval between injection of forward ant packets at each node is set to be constant. The time between the injection of forward ants for a certain node (i, j) is given by $T_{fwd,i,j}$. Thus equation 5.2 can be simplified as below

$$T_{LT,i,j} = F_{ij} T_{fwd,i,j} + T_{fwds,i,j} + T_{bk,i,j} \tag{5.2}$$

Equation 5.2 gives the learning time to discover a path from node i to node j. For a partition comprising of N number of nodes, the learning time of one node is given by equation 5.3. Since the discovery of paths are done in parallel, the learning time for one node ends when all the paths are discovered from this node to every other node and is equal to the maximum of them as shown in equation 5.3. Thus the total learning time for all the nodes in a partition is equal to the node which has the longest learning time and is provided by equation 5.4.

$$T_{LT,i} = \max\{T_{LT,i,j}, \forall\, j \in 1, ..., N\}, i \neq j \tag{5.3}$$

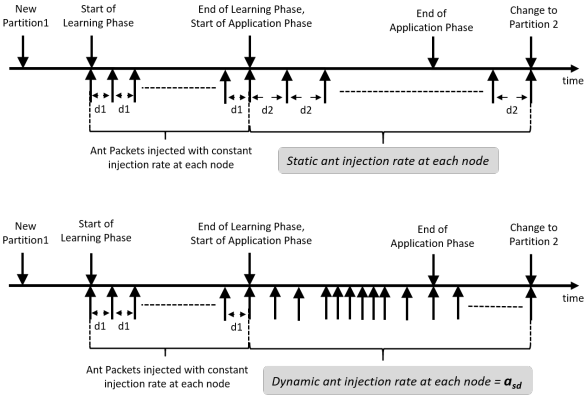


Figure 5.5: Overview of DREAM framework[13]

$$T_{LT} = \max\{T_{LT,i}, \forall i \in 0, 1, ..., N\} \tag{5.4}$$

5.2.4 Application Phase Analysis

During the start of Application Phase (AP), data packets are injected into the network as well as special ant packets. Depending on the criticality of the packets either the deterministic routing is used (GS for critical packets) or ACO based routing is used (BE for non-critical packets). Special packets like forward ant packets are injected at each node to discover new and optimal paths. In the learning phase, forward ants are injected at each node at a constant rate (for example every few cycles to ensure low overhead). Once a node discovers a path to a certain destination, it stops injecting forward ants to that destination to decrease traffic and thereby decreasing learning phase time overhead. The paths discovered at the end of this phase can be sub-optimal as the learning has just started and data traffic is not present. Better paths are discovered due to the combination of long term and local traffic information. Therefore, during the application phase, forward ants are injected at all nodes once again, to discover

better paths due to the traffic fluctuations. Injection rate of ant packets during the application phase can be static or dynamic as shown in Figure 5.5. In the static technique, each node injects ant packets at the same constant rate and is not dependent on the rate of data packets injected. In the dynamic technique, ant packet injection rate of a node is dependent on its data packet injection rate. Consider a forward ant injection rate at a node s to destination d is a_{sd} as shown in Equation 5.5. Where c is a constant and IR_{sd} is the injection rate of the data packets from the node s to d. The injection rate IR_{sd} can be static or dynamic depending on the applications.

$$a_{sd} = c * IR_{sd} \tag{5.5}$$

5.3 Exploration vs Exploitation of Paths

According to the original ACO algorithm [101], which is used as a basis here, when the value of α in Equation 4.4 is close to 1, the contribution of the ants collective learning is dumped. And the system closely follows local traffic fluctuations resulting in possible large oscillations in performance. On the other hand, an α close to 0 results in the decision completely dependent on the long-term ant learning process. This can make the algorithm unable to follow quick variations in traffic patterns. Thus in both cases, the system is not expected to perform satisfactorily. In both the cases, the number of ants generated at each node is expected to play a critical role According to [101]. Since a large number of ants can aid in overcoming the negative effects of α close to 0. The large number of ants can also mitigate the effects when α is close to 1 by creating an over-reactive behaviour. From the experiments carried out by the authors in [101], a good and robust choice of an α is in the range between 0.2 and 0.5

Routing tables maintained at each router store pheromone values which indicate goodness of a port for a particular destination. A path is selected with a probability, which is a combination of this pheromone value and link attractiveness

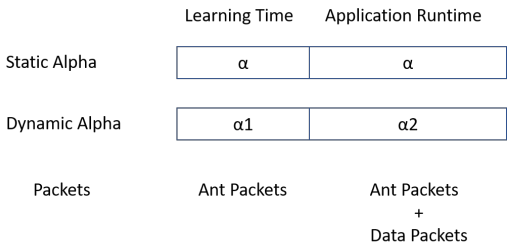


Figure 5.6: Overview of the static and dynamic α technique [109]

as explained in Section 4.1.3. A constant α which represents the trade-off between exploration of new paths and exploitation of the ones already discovered is used when computing the probabilities. Instead of treating the α value as static, we propose dynamically changing the α value over the learning and application phase. The motivation here is to investigate the influence of α over the time it takes to discover paths in a region. Using this dynamic α technique we aim to decrease the discovery time overhead while maintaining comparable throughput and latency. This work was published in [109] and an overview of the concept is described in the next section.

5.3.1 Decreasing Learning Time Overhead

We aim to investigate the influence of α over time taken to discover paths in a partition and dynamically change α to reduce the learning time overhead. We propose the runtime adaptive α technique to decrease learning time while maintaining overall latency and throughput. An overview of the runtime adaptive α technique is shown in Figure 5.6. The learning time is used to discover paths in the given topology in a region. This path information is used by the data packets when the learning time ends and application starts running. During the learning time, only ant packets are generated by each node in a region to discover at least one path between every pair of nodes. During this time the pheromone tables at each router are also being updated. This learning time is used by the ACO based routing to discover routes within a certain partition. A certain value of $\alpha 1$ is set

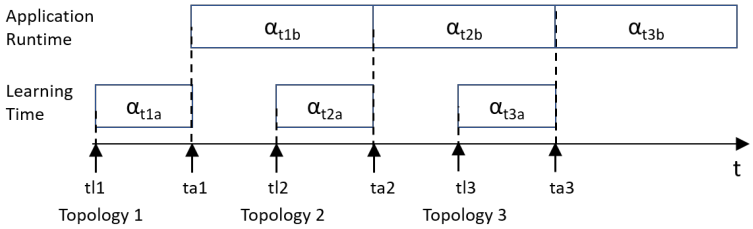


Figure 5.7: Example of a dynamic α technique for multiple topology changes [109]

as an initial value during this time period. When at least one successful path between every pair of nodes is discovered, the learning time ends. The $\alpha 1$ value is now changed to $\alpha 2$. Application data packets are now being generated along with ant packets as the application has now started running. Generation of ant packets continue, to find better paths than originally discovered, now taking into account the varying data traffic patterns as well.

As a reference, the static technique where α remains constant and is the same for both the learning time and application runtime is also shown in the Figure 5.6. We make a differentiation between learning time and application time to ensure that when application packets start using the network, at least one feasible path is already available between every pair of nodes to avoid packet loss due to lack of feasible path. To set the initial value of $\alpha 1$, we conduct tests to determine which α value is suitable to achieve the least learning time. The α value which performs better when there is a high amount of data traffic especially at network saturation is selected for $\alpha 2$. Using this technique we aim to achieve the lowest learning time, while ensuring the latency and throughput do not suffer.

If a partition shape changes after an application has started running, path information for the resulting new topology needs to be computed. Partition shapes can change when a higher priority application sharing the NoC consumes additional network resources for example. An extension of the dynamic α technique when multiple topology changes occur is shown in Figure 5.7. Initially $\alpha 1$ and $\alpha 2$ for topology 1 are α_{t1a} and α_{t1b} respectively. After a period of time, when topology changes, the second learning time t_{l2} commences to discover paths in topology 2.

This second learning time occurs in parallel to avoid halting the application. The $\alpha 1$ and $\alpha 2$ for topology 2 in the Figure are α_{t2a} and α_{t2b} respectively.

5.4 Ensuring Quality of Service

Quality of Service (QoS) in general is the overall performance of the network. It can include bandwidth, latency and throughput for example. In the NoC context, QoS can be the latency or throughput experienced by a component utilizing the interconnect [20]. Guaranteeing QoS is the ability of the network to promise certain throughput and latency. This is especially important for applications having real-time, safety or security requirements. If throughput and latency can be guaranteed by the network, it is referred to as Guaranteed Service (GS) [20]. To enable guaranteed service communication typically network resources are reserved and prioritization techniques are used to ensure data traffic meets the requirements. A challenge in adaptive NoCs is ensuring latency and throughput guarantee, especially when dynamic routing is used. There is benefit when utilizing run-time adaptations, but providing adaptative techniques and in parallel maintaining guarantees is still an open research problem as mentioned in [9].

In this work, we present DREAM routing and a framework to execute it in mixed critical scenario.We had restricted that only non-critical traffic use ACO based routing. If latency and throughput guarantees can be provided when using the ACO based routing, critical applications can also benefit from the flexibility of such an adaptive routing. The base router used supports both GS and BE communication [20]. When using BE, data to be transmitted from source to destination is split into packets and transmitted. It is packet based communication and the packets can take different routes. In BE communication, packets obtain best-effort service, the possible data rates and latencies cannot be clearly provided at design time as the results depend on the current traffic load and the traffic load in a distributed system is difficult to determine [20]. The base router when using GS communication can provide hard guarantees. When using GS communication, network resources like virtual channels and time slots are reserved from source to destination. Efficient

resource sharing using virtual channels are used by allocating virtual channels exclusively for GS communication. The base router supports reconfigurable bandwidth. This is achieved by reserving a certain number of timeslots when data is transmitted. The number of timeslots to be reserved are specified in the header of a GS packet. This enables not just GS communication but also bandwidth can be configurable at runtime.

5.4.1 Latency and Bandwidth Analysis of Base Router

The base router used [20], can provide hard guarantees regarding throughput and latency. Since resources are reserved exclusively for GS communication and thus isolating it from other data flow. For an established GS connection, latency and throughput can be guaranteed and hard boundaries can be provided. The worst case bandwidth when using GS service is provided below. The bandwidth for a given service level $SL \in \{0, 1, ..., SL_{max}\}$ is provided in Equation 5.6.

$$BW_{GS}(SL) = \frac{SL}{TS_{util,max}} \cdot BW_{link} \tag{5.6}$$

where $TS_{util,max}$ is provided by Equation 5.7. BW_{link} is the bandwidth of the physical link.

$$TS_{util,max} = \max\{TS_{util,i}, \forall i \in \{0, 1, ..., k\}\} \tag{5.7}$$

In Equation 5.7, $TS_{util,i}$ is the current port utilization during transmission. Where i represents the ports 0 to k used by an established GS connection. The transmission latency L_{GS} for a packet for a GS communication using a certain SL is provided by Equation 5.8.

$$L_{GS}(SL) = (TS_{util,max} - SL + 1) \cdot (H \cdot L_{pipeline} + S_{pkt} - 1) \tag{5.8}$$

Where H depicts the number of hops of the packet from source to destination. $L_{pipeline}$ is the flit processing delay per hop. It depends on the number of pipeline stages taken by the body flit in a router. The packet size in flits is provided by S_{pkt}. To compute the worst case latency, since $TS_{util,max}$ can change at runtime, in 5.8, the value of $TS_{util,max}$ is replaced by TS_{total} which is known at design time. All other parameters like SL,$L_{pipeline}$, S_{pkt} and H are known at design time. Deterministic routing is used, where the number of hops H between source and destination can be known at design time.

Let us take an example used in [20] for reference to determine the worst case latency for a an example GS communication. Consider a GS communication flow established between a source and destination node. The header flit reserves virtual channels, router resources and channels along the path. The routing is a static XY routing. Let us compute $L_{GS}(SL)$ for a SL of 4. This implies Service Level is set to 4, so 4 out of the maximum available time slots (which is 8 in the design) are used for communication. Let us consider in this example, the hop count between the nodes is 8. So H is 8. $TS_{util,max}$ is the maximum number of time slots available, which is 8 for the base router design.

The base router is a pipelined design with 5 stages. An overview of the functions at each stage is shown in Table 6.1. Minimum 2 stages are required for functioning of the router and the other three are optional. Depending on the type of flit the number of processing cycles in a router is different. For example the stage 2 and 3 processes only Header flits. In computing the worst case latency, $L_{pipeline}$ is the flit processing delay per hop for only the body flits. In our example let us consider a 5 stage pipeline. Here, the processing delay per router for the header flit is 5 cycles and for body flit is 3 cycles. So $L_{pipeline}$ here is 3. The packet size is reflected by S_{pkt}, and let us set it to 16 in this example. Now $L_{GS}(SL)$ for this example is 195 cycles. Let us focus on the routing aspect. Its is static XY routing, which is deterministic. Hop count between a source and destination can be determined at design time. In the next section, we discus how one can potentially ensure guarantees when the ACO based routing is used.

5.4.2 Worst Case Analysis of DREAM Router

Using the DREAM router in which the ACO based routing is integrated, the paths which the packet can take between nodes can change when better paths are discovered. This makes determining the exact hop count for a given pair of nodes challenging at design time. Now, how can we ensure latency and throughput guarantees using DREAM routing?

One method to ensure guarantees is to assign a threshold to the hop count allowed for a given pair of nodes. Let us have a look at the latency analysis. The value which is not deterministic at design time is the number of hops H. If we assign a threshold to this hop count at design time, the worst case latency can be determined at design time. Threshold can be assigned per pair of nodes or assigned the same for all pair of nodes. Let us analyse for when a threshold is assigned per pair of nodes. A Hop Count (HC) is the number of hops a packet takes to travel from source node to destination node. Minimal Hop Count (MHC) is the smallest hop count connecting a pair of nodes. To ensure a low worst case latency estimate, one can set the hop count H in Equation 5.8 to be the minimal hop count. If the topology is known at design time, this is one option for assigning threshold.

Here we aim to improve network flexibility, especially when aiding in spatial isolation for mixed criticality systems. When partitions are created, let us consider the topology within the partition is unknown at design time. In such cases the threshold can be set to higher number for example increased by 2 hops. This allows for non-minimal paths which can be beneficial. For example when ant packets discover better paths around a certain hotspot or failed node, taking a non-minimal path to reach the destination. Or due to the boundary of the partition, non-minimal paths need to be taken. The drawback is that the worst case latency will increase accordingly. An overview of how the worst case latency increases with the increase in hop threshold is illustrated in Figure. 5.8.

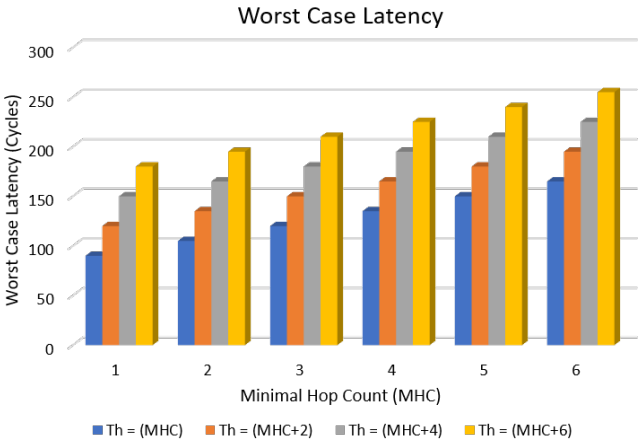


Figure 5.8: Variation of worst case latency for different hop thresholds

5.5 Multi-Layered DREAM NoC

In the last chapter in Section 4.5, we had presented a multi-layered NoC topology with a supporting adaptive congestion and criticality aware routing to decouple mixed critical traffic. Critical packets were routed across shorter paths and non-critical packets were directed over short or longer paths depending on local network information. Here we propose to utilize the DREAM router which runs the reinforcement learning enabled routing in such a multi-layered NoC to transport non-critical traffic. The ACO based algorithm is extended for multi-layered NoCs and this work is published in [106]. Flexibility can be increased if a topology agnostic routing can be utilized.

The multi-layered topology under test in which the ACO based routing is implemented was presented in the previous chapter in Figure 4.15. The network is illustrated again in Figure 5.9 and the variation of port count in the routers are also presented. To refresh, it comprises of 25 nodes interconnected using three network layers. The base layer referred to in the Figure as Layer 3 is a 5x5 mesh layer. The second layer interconnects alternate nodes forming a 3x3 mesh

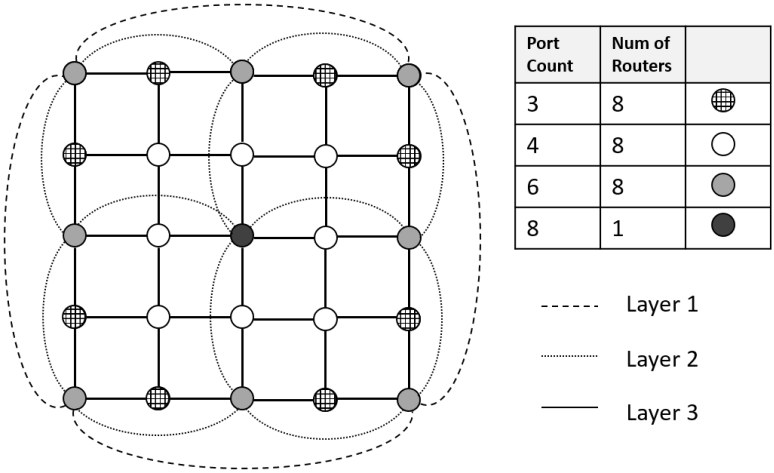


Figure 5.9: Variation in router count and potential of path diversity in a 5x5, three layer network [106]

network. The top layer interconnect alternate nodes of layer 2 forming a 2x2 layer. Such a topology allows multiple paths between pairs of nodes with different hop counts.

Here, we exploit the diversity of paths available, by implementing the DREAM routing to transmit packets.In such topologies, the routers can have varying degree of port counts as illustrated in Figure 5.9. Port counts of routers range from 3 to 8 as shown This results in routers with higher port counts facing more traffic which contributes to formation of hotspots. This can have a detrimental effect on latency and throughput. A fully adaptive and dynamic DREAM routing, continuously discovers better paths between nodes.

Here, we implement such a routing mechanism on the multi-layered topologies as it inherently exploits the potential of path diversity in such networks. Packets are transmitted based on local network state and past history of selected paths. The routing table located at each router is updated in a distributed manner by the special ant packets when new paths are discovered. This enables the algorithm for example to navigate around hotspots and aid in load balancing. Due to the

diversity of paths available between nodes, potentially better paths are explored by special ant packets over time.

For critical packets, static routing like the XY is used to ensure guarantees and for non-critical packets dynamic routing like the ACO based routing is used. To ensure guarantees and low latency critical packets travel on static paths comprising of shortest hop counts. This can result in certain routers becoming hot spots at high traffic scenarios. Especially when the communication links are also being shared by non-critical packets. To reduce this inter application influence, the non-critical packets are routed on alternate paths discovered using the DREAM routing.

Routing table at each DREAM router is built over time using the special ant packets. There is a learning phase overhead to build the routing tables before the application can use the network. If the topology is fixed, an option is to run this learning phase and build preliminary routing tables offline, at design time. Here in this work, we aim to keep the routing scheme flexible to also support networks having configurable topologies. Where topologies may not be fixed at design time. To ensure this overhead is low, once preliminary tables are built where at least one path is available between each pair of nodes in the network, the application can start its runtime and start using the network to transmit the packets. In the example here, the network is the 5x5, three layer topology Special ant packets are injected along with application packets, to discover better routes over time depending on application data patterns.

5.6 Summary

In this chapter, we presented the DREAM NoC which uses the reinforcement learning enabled routing based on ACO. We also presented the supporting framework which comprised of dedicated learning phase and application phase to aid in flexible, adaptive, runtime spatial isolation in mixed criticality systems. The analysis of learning time, which is dedicated to the discovery of paths in the given

network and building of preliminary routing tables was presented. Techniques by which this learning time overhead can be decreased was proposed. The latency and bandwidth analysis of the DREAM router along with the worst case analysis was discussed. We also extended the DREAM routers to support multi-layered NoCs.

6 NoC Architecture Realization

This chapter describes the architecture of the base NoC router, DREAM NoC and multi-layered NoC. This includes implementation description of adaptive routing algorithm concepts presented in Chapters 4 and 5. Additionally, design constraints, modifications done to make the DREAM routing algorithm implementation more hardware friendly and custom functions implemented is described in this chapter.

6.1 DREAM Router Architecture

An overview of the DREAM router architecture and its functionalities is described in this section. The base router is based on [111] and an overview of its working is presented in Section 2.3.2. A basic architecture overview is shown in Figure 6.1.

6.1.1 Base Router Features

An overview of the basic functioning of the router is as follows. The router utilizes packet switching and the concept of virtual channels. The packets are transported using wormhole switching technique for efficient buffer utilization. Arriving packets are stored in the input buffers. In the design, there are four buffers per input port, each representing a virtual channel. The number of input ports can be set at design time. For a regular 2D mesh topology the number of ports is 5. Four ports in each of the cardinal directions (N,S,E,W) and a local port(L). When the

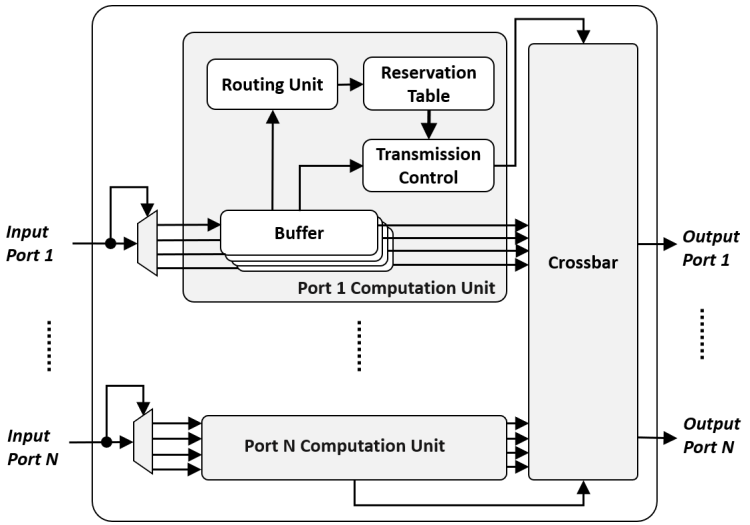


Figure 6.1: Base Router Architecture.

Ctrl Flag(1)	Head Flit	Ctrl Flag(0)	Body Flit	...	Ctrl Flag(0)	Body Flit	Ctrl Flag(1)	Tail Flit

Figure 6.2: Overview of packet structure comprising of head flit, one or more body flits and a tail flit.

packet arrives at the input buffer, information like source and destination addresses are sent to the Routing Unit (RU). The output port is determined according to a routing algorithm in the routing unit. A reservation request to the determined output port is made in the Reservation Table (RT): The reservation table selects a certain VC at the requested output port and reserves it for the requesting input port. If all VCs are occupied, packet waits in the input buffers until a VC is available. A Transmission Control unit, controls access to the output port and arbitrates between the requests reserved to reserve the output port. A round-robin scheduling scheme is implemented in the transmission control. The crossbar is switched on by the Transmission Control unit and the data in a selected input port is forwarded to its requested output port accordingly. To ensure there is enough space in the input buffers of the downstream router, a credit-based flow control is used.

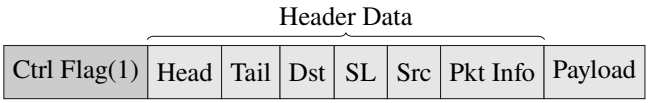


Figure 6.3: Format of a basic control flit appended with a control flag. A control flit can be a head flit, tail flit or a single flit.

As the router design uses wormhole-switching, packets are divided into equal size flits. A packet usually comprises of a Head flit, one or more Body flits and a Tail flit. An overview of the basic packet format is shown in Figure 6.2. Each flit is transmitted along with a control flag prefix to indicate if it is a body flit or a control flit. A control flit can be a Head flit, a Tail flit or a single flit. A single flit contains information of Head, Tail and body in a single flit. Addresses and other relevant packet information needed to determine paths are present in the Head flit and a control flag aids the router to process the different packet flits accordingly. Format of a control flit and a data flit is illustrated in Figure 6.9 and Figure 6.10 respectively. The size of the flit depends on the link width which can be configured at design time. A Head flit is indicated by setting the respective field to 1 and Tail field to 0. Similarly for a Tail flit, the respective field is set to 1 and the Head field is set to 0. The source and destination addresses are stored as X and Y co-ordinates. Packet Information field contains information which might be required and depends on the routing algorithm used.

The base router is hybrid as it supports both GS and BE communication. The GS communication [111] provides different Service Levels (SL) and higher the value of SL, higher is the bandwidth guarantees provided and with lower latency. Applications can set the SL value in the packet header according to their requirements. SL value indicates if it is a BE or a GS packet. A zero value indicates a BE packet and a non-zero value indicates a GS packet where the value of SL indicates the service level requirement in terms of bandwidth allocation. The SL value represents the number of time slots which can be reserved to transmit data between one router to another.

To improve timing and clock frequency, a pipeline model is utilized. An overview of the pipeline stages is shown in Table 6.1. Not all stages are required and the

Figure 6.4: Format of data flit comprising of the payload appended with a control flag

Table 6.1: Pipeline Stages of Base Router

Pipeline Stage	Description	Type of Flit	Note
1.Input Buffer	Requires one cycle	All(H/T/B)	Required
2.Routing	Route/output port computation and Arbitration	H	Optional
3.Reservation	Add entry in the reservation table	H	Required
4.Scheduling	Scheduling using arbitration unit in transmission control unit	All(H/T/B)	Optional
5.Output Register	Can be enabled to relax timing of the links	All(H/T/B)	Optional

number of stages can be configured at design time. At least two stages are required for the functioning of the base router and the other three are optional. Depending on the type of flit the number of processing cycles in a router is different. Here we implemented a XY routing mechanism in the base router. To support multicast and broadcast communication, we developed and implemented the block-based multicast routing concept described in Section 4.4 and the results was published in [100]. This Block-Based multicast routing is implemented in the Routing Unit and changes were made to the crossbar to support packet duplication when required.

In GS communication in the base router, the header flit reserves the router resources from node to node as a chain of virtual channels. Body flits are used to transmit data and tail flit releases the resources. The end-to-end GS communication is established by exclusively reserving resources and is isolated from other communication flows in the network. Thus hard guarantees in latency and throughput can be provided using GS communication.In BE communication all the flits of the packet, like header flit,body flits and tail flits are injected contiguously into the network and there is no reservation of communication resources

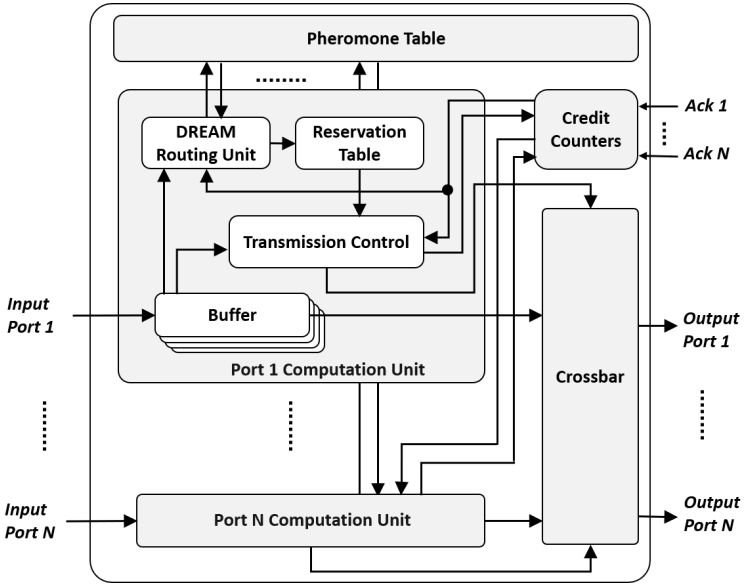


Figure 6.5: An overview of the DREAM router

by the header flit. All packets therefore obtain BE service and the latency and throughput depend on the current network load.

6.1.2 DREAM Routing Unit

The DREAM routing is a reinforcement learning enabled routing algorithm targeting mixed criticality systems and is based on AntNet routing algorithm. As described in section 4.1.1, we aim to implement certain aspects of the AntNet routing algorithm which is based on ACO in the NoCs and target mixed criticality systems. Preliminary investigations into implementing the AntNet routing in a NoC router was conducted as part of a supervised student's work [37]. This included investigating modifications to the original AntNet routing to simplify for hardware friendly implementations.

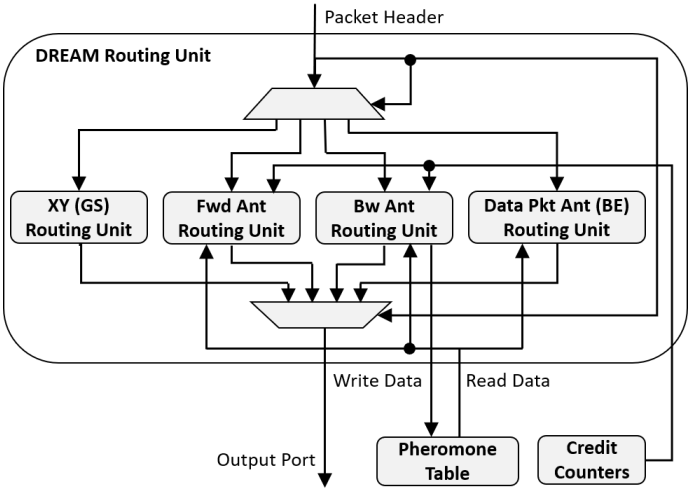


Figure 6.6: An overview of the DREAM routing unit

An overview of the DREAM router is illustrated in Figure 6.5. Each computation unit comprises of a routing unit implementing the DREAM routing algorithm and accesses the pheromone table accordingly. Relevant router features like the credit counters is shown here which keeps track of the input buffer status of the downstream router via the acknowledgement signals received. This buffer status information is used by the transmission control to determine when the buffer is free to transmit packet and is also used by the DREAM control unit to determine better paths. The number of input ports depend on the chosen topology. For a mesh network, there are 6 ports: four in the cardinal directions (N,S,E,W), a local port (L) and a drop port. The four cardinal output ports are connected to the neighbouring routers and the local port is connected to the local processing tile via a Network Adapter. A drop port is built into the router is drop any forward ant packets which were not successful in finding a suitable output port.

A more detailed architecture of the DREAM routing unit is shown in Figure 6.6. When a packet arrives at the input router and is sorted into an input buffer, the header information is accessed by the DREAM routing unit. A suitable routing

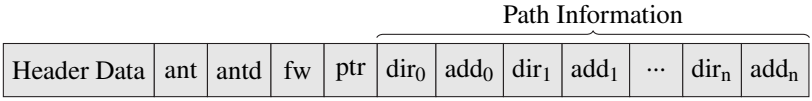


Figure 6.7: Format of a forward ant flit

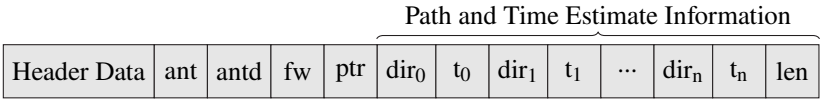


Figure 6.8: Format of a backward ant flit

algorithm is utilized based on the criticality of the packet. For a packet requiring GS, a deterministic routing like XY is utilized to compute the output port. For a packet requiring BE service, output port is computed using the DREAM routing. DREAM routing comprises of processing the forward ant packets, backward ant packets and the data packets.

6.1.3 Packet Format

Forward and backward ant packet sizes are restricted to a single flit to reduce router processing delay. An illustration of a forward ant flit is illustrated in Figure 6.7. Along with the base Header data it contains fields to indicate the type of ant flit and stores the path information. The length of the path is restricted by a certain upper bound to the number of hops allowed when discovering a forward path. This threshold value can be set at design time. This also aids in avoiding livelocks, as packets are dropped if the number of hops exceeds the value set. Livelocks are scenarios where packets travel continuously in a network without reaching its destination. An overview of the descriptions of the fields is illustrated in Table 6.2. In our design, once the forward ant flit reaches the destination router, it is converted into a backward flit and sent to the local port connecting a test bench. The test bench implemented, receives this backward ant flit and injects it back into the network. The backward ant flit format is illustrated in Figure 6.8. In the backward ant flit, the source and destination addresses are swapped and the path information like the directions is retained along with the total length of the

Figure 6.9: Format of a DREAM control flit appended with a control flag. A control flit can be a head flit, tail flit or a single flit

Figure 6.10: Format of data flit comprising of the payload appended with a control flag

path. Direction information is enough for the backward ant to travel back to the source router. The address fields in the forward ant flit is not needed any more and its fields are reused to store link travel time information per hop. The backward ants when travelling back to the source node, on its path, stores the estimated link travel time for each hop and stores it in the respective field. Reusing the fields in such a manner helps to maintain same flit length for forward and backward flits. Data packets comprise of a head flit, one or more body flits and a tail flit. Investigations into preliminary design of packet formats was conducted as part of the supervised student's work [37].

6.2 Design constraints

In this section, we describe how certain complex computations was handled to make the implementation hardware friendly. Preliminary investigations was conducted as part of the supervised student's work [37] and an overview is provided in this section and in section 6.3.

6.2.1 Router local memory

Pheromone values are represented by 4 bit fixed point numbers and is based on the research by authors in [112]. This is because, pheromone values are always in the range of $[0, 1)$, an integer part is not required and all bits can represent the fractional part. The router local memory stores the pheromone table and the traffic model.

Table 6.2: Fields of forward and backward ant flit

Field	Description
Header Data	Comprising of base packet information like source and destination addresses, as mentioned in Figure 6.9
ant	if set to 1, indicates a forward ant, backward ant or a data packet following DREAM routing
antd	data ant field, if set to 1, indicates a data packet to be routed using DREAM routing. If set to 0, and ant is set to 1, indicates a forward or backward ant flit
fw	if set to 0, indicates a forward ant flit. If set to 1, indicates a backward ant flit
ptr	for forward ant flit, it is to keep track of free cell to add address when discovering paths. For backward ant flit, it is to keep track of directions to travel back to the source
dir_0-dir_n	forward ant flit stores directions is takes, so the backward ant can travel back to the source
$addr_0$-$addr_n$	forward ant flit stores addresses of nodes it visits. Aids in avoiding loops.
T_0-T_n	backward ant flit stores link travel time estimate between nodes
len	length of the total path
payload	In head flit of a data packet, stores packet ID information. In data flit, stores the data to be transmitted

Each DREAM routing unit performs a read access this router's local memory when a forward and or a data packet arrives. Ant learning is part of the backward ant routing unit and produces a read-modify-wrote cycle to this memory when required. If the number of input ports is N (including the local port), the local memory storing the pheromone table needs to support N concurrent read accesses per clock cycle and a read-modify-write cycle for the destination of interest. The memory needs to also support N-1 concurrent writes, as backward ant flits can arrive on all external directions except the local link. Write collisions have to be tolerated. This ant memory comprising of the pheromone table is implemented

as synchronous RAM. Thus the read data is only valid for one cycle after the address. Due to this, the optional routing pipeline stage 2 in the base router is now mandatory. This is done so that in this cycle, the read address can be generated from the destination address in the packet header, and the resulting information is used route the packet in the next cycle This modification results in 3 mandatory pipeline stages for the DREAM router. The pheromone table at each router stores pheromones of all source-destination pairs. If a maximum possible partition size is known at design time, then the size of the table can be set accordingly. Initial values of the table is set based on a superposition of XY and YX routing for a mesh based topology to allow for path diversity.

The values of link queues is derived from the credit counters which keep track of the number of flits in the input buffers (virtual channels) in the downstream routers. This gives a measure of the number of bits waiting in the buffers, which was intended by the AntNet algorithm as well. The DREAM routing unit has to process an entire ant packet before an output port can be determined. In our design, the packet is of one flit size of 128bit. To decrease the link width, an ant packet can be split into multiple flits but this can increase routing processing delay. To simplify integration, reading and writing processes into the local memory, ant packets are reduced to a single flit of size 128bit.

6.2.2 Overview of Modifications

An overview of modifications made to the AntNet routing algorithm to simplify it to make to keep the hardware complexity manageable is explained in this section. Sub-path updating is not performed as it requires an additional read-modify-write cycle to the local memory for each sub-path. Second modification is the correction term in the computation of $\tilde{r}$ in Equation 4.12 is excluded which implies setting the parameters $c_1 = 1$ and $c_2 = 0$. This is done in our case as the correction term has a little impact on the resulting reinforcement factor most of the time but is responsible for expensive circuits. This simplification also eliminates the mean and variance from the traffic model. As these two moments would require higher

Table 6.3: Overview of AntNet's parameters and the values assigned in our design [37]

Parameter	Useful Domain	AntNet	Here	Notes
α	$[0.2, 0.5]$	0.2	$[0, 1]$	exploration vs. exploitation, values beyond the suggested useful domain is investigated to decrease the learning time overhead
ϵ	> 1	1.4	2	data exponent
$\hat{w}$	$c \cdot 5/\eta$	$c = 0.3$	15	window length. Maximum value is 15, can be stored in 4 bits
a	> 0	5	5	squash factor

precision fixed point formats, this makes storage in the router memory structure and updation logic less complex.

The AntNet algorithm has a number of parameters which influence the behaviour of the algorithm and the its authors provide a useful domain for these parameters along with the values used for their experiments in [36]. The values used in our design are derived from these suggestions and they are rounded to the nearest power of two to enable less expensive hardware implementation using bit shifts. Table 6.3 gives an overview of these parameters.

6.3 Functioning

6.3.1 Forward and Backward Ant Routing

Forward ant flits are injected at each node in the network. When a forward ant flit is routed, a list of all possible directions is first determined. Probabilities of the output ports are then calculated based on Equation 4.4 and a direction is selected randomly.The routing decision is stored in the ants memory along with the local

address as described in Section 6.1.3. When considering possible directions to route the forward packet, physical links which do not lead to an already visited node is considered. If no such path exists, the ant flit is dropped. On the way to the destination, if a loop is detected, the forward ant flit is dropped. At the destination router, the forward ant flit is converted to a backward ant flit and it retraces back to the source node based on the stored path information.

Backward ant flit inherits the forward ant's memory, and on its way back to the source stores the link traversal time for each hop. Address space is reused to store the traversal time, as this information is not required any more. The conversion of the forward ant flit to the backward ant flit occurs in the destination router of the forward flit. The packet then is ejected via an output port and injected back into the router to start the backward journey. To support this feature the base router was modified to support the ejection of packet via a routing port through which it has entered.

6.3.2 Ant Learning

When the backward ant arrives at a router, along with the processing of the backward ant flit the ant learning is also executed. Due to the modifications to the original AntNet algorithm to make it hardware friendly the traffic model procedure is simplified as described in Section 6.2.2. Due to the simplifications which include setting $c_1 = 1$ and $c_2 = 0$ and consequently eliminating mean and variance from the traffic model, only the non-sliding window remains a part of the traffic model. Refer to equations 4.12 and 4.13. Now, W is reset to the average of best and current travel time to approximate the longer term mean:

$$W' = \frac{T + W}{2}. \tag{6.1}$$

The reinforcement factor now gets simplified to

$$r = s_n\left(\frac{W}{T}\right). \tag{6.2}$$

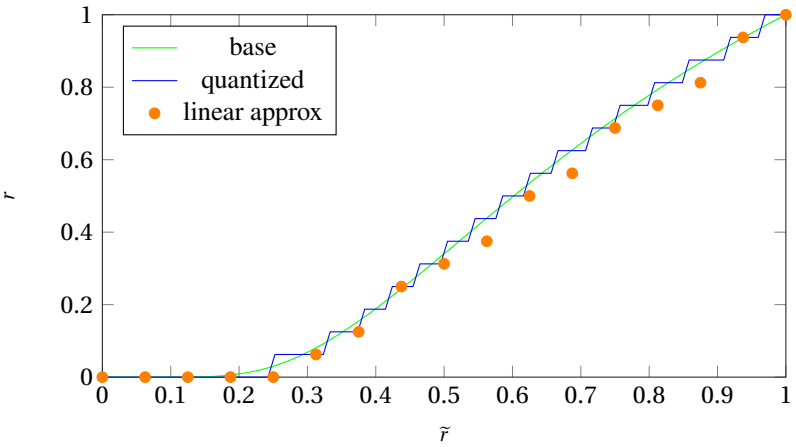


Figure 6.11: Squash function and approximations for $|\mathcal{N}_k| = 4$ [37]

The squash function is used to emphasis good results by squashing the reinforcement factor as described in Section 4.1.4. The squash function as shown in Equation 4.14 comprises of exponentials and divisions which increases hardware complexity. However after quantization to 4 bit, s looks almost linear. The approximation is illustrated in Figure 6.11 and is also compared to the original squash function. This motivates replacing the original definition with a linear approximation.

6.3.3 Data Packet Ant Routing

Data packets which uses DREAM routing is routed according to the ant data routing policy described in Section 4.1.3. The routing probabilities are derived based on the pheromone values using equation 4.5. The parameter, data exponent is set to $\eta = 2$ as described in Table 6.3. The integer power results in less expensive hardware. This also results in increased emphasis to good paths when compared to AntNet routing due to the higher exponent value. Forward ant and backward ant flits are modified during routing. For data packets, only the head flit information is sent to the routing unit for processing and the rest of the packet is stored in the

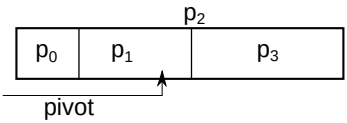


Figure 6.12: Random port selection [37]

input buffer. This results in the data path being spilt before the routing unit, and a multiplexer and additional registers are utilized to reintroduce the modified flit into the data path. Data packets requesting for BE communication is routed using DREAM routing and GS communication is routed using deterministic routing. For a mesh based topology one of the algorithms we implemented is XY routing algorithm.

6.3.4 Random Port Selection

When routing forward ant flits and data packets, the output port is selected randomly with a probability based on the pheromone value. This is implemented by adding the probability values to an accumulator. When a probability value is added to the accumulator, the current sum is compared to a pivot register that is random and uniformly distributed. If the accumulator sum is greater than the pivot, the recently added probability is selected. An example of the selection process is shown in Figure 6.12. The selections are only made with the correct probability if the sum of all the probability values add up to 1. Thus a normalization step is required. A pseudo random number for the pivot is generated using a linear feedback shift register. A dedicated pseudo random number generator is implemented in each DREAM routing unit with a unique seed which is derived from the node address and port number. This randomness in selecting an output port does not result in same routing decisions every time, which could lead to congestion of ports which is now avoided.

6.3.5 Division and Normalization

Investigations to evaluate suitable division and normalization techniques was conducted as part of supervised student work [37] and a summary of the results is presented here. The base router used is a pipelined design and to effeciently integrate the DREAM routing scheme into the architecture, we aim to execute the routing decision in a single cycle. Due to this, the division needs to be combinatorial. Binary long division scales poorly with precision [113], but it is combinatorial and can be effective for low precision numbers which is the case in our design. The initial synthesis conducted in [37] revealed that this approach reaches 77 MHz for the normalization of four pheromones. The circuit consumed 560 LUTs. An alternative method was that a division can be converted into a unary inversion followed by a multiplication. The inversion could be performed with a look-up table which holds all possible $1/q$ values and a division can then be expressed by

$$\frac{p}{q} = p \cdot (1/q). \tag{6.3}$$

This circuit reaches 300 MHz and uses only 47 LUTs and 4 DSP slices which accelerate the multiplication. Thus the described alternate version was chosen to implement the divisions.

6.3.6 Traffic Isolation Implementation

The DREAM router supports GS and BE communication as explained in Section 6.1.2. Non-critical applications use BE service where DREAM routing is used to route packets and critical applications use GS communication which is routed on deterministic paths using XY routing. The partition shape and thus the topology occupied by critical application is usually known at design time and guarantees need to be computed at design time. When a critical application arrives and occupies resources, the region information is communicated to the border routers to stop accepting any packets from nodes outside this region. Thus if an exploring forward ant packet attempts to enter this region it is dropped, thus discouraging

future ants attempting to enter the critical region. In general it is sufficient for the pheromone table at each router to store route information within its own partition. Thus the table size can be set to maximum number of nodes possible in a region and not the total number of nodes in the entire network to improve scalability.

6.4 Deadlock and Livelock Avoidance

In the design, deadlock and livelock avoidance mechanism is implemented as follows. A method to avoid deadlocks is by restricting turns in a turn model [14]. In turn models, possible deadlock cycles are defined using the turns which the packets can potentially take. In a 2D mesh topology, there are eight turns which can be combined to create two abstract cycles in clockwise and anti-clockwise directions and is shown in Figure 2.11. In our design, one of the turns in each cycle is eliminated to avoid cyclic dependencies and thus avoiding deadlocks. When a forward ant packet is discovering a new path, at each router, the DREAM routing unit determines a suitable output port. The routing unit uses the information of the input port it arrived on and eliminates output ports which can lead to a forbidden turn. Over time due to the reinforcement learning aspect of the routing mechanism such turns are not rewarded and packets seek alternate paths. Message dependent deadlocks are avoided by restricting certain packets to an exclusive virtual channel. For example, memory based traffic data like request and response packets can cause message depended deadlock even if the routing algorithm used is deadlock free (XY routing for example). To avoid such cases, in the design request packets are restricted to 3 virtual channels (example vc1, vc2, vc3) and response packets are restricted to travel in its on fourth virtual channel (example vc0). Due to this restriction the request and response packets do not share the same virtual channel and message-dependent deadlocks caused due to these packets can be avoided.

Livelock situations are when a packet is travelling in the network without reaching its destination and can have a detrimental effect on performance due to increased traffic. Such cases can be avoided by dropping the forward packet after it reaches

a certain number of hop count value. The threshold for the hop count can be assigned at design time and can depend on the topology. The forward ant packets store the path information its memory, and using this information a hop counter is used to determine the total number of hops travelled by the ant packet. Is the forward ant packet meets a certain threshold the packet is dropped and such paths are not rewarded and results in the path being avoided by future ant packets. In our design, for a mesh based topology, we assign the threshold to greater or equal to the diameter of the network to enable non-minimal paths. Diameter is the shortest hop count between the farthest nodes of a network. To support non-minimal paths for irregular topologies and to support longer paths we set this threshold to twice the diameter for mesh based networks. Such non-minimal paths can be advantageous to route around hot spots or faulty routers or links.

6.5 Multi-Layered NoC Architecture

We have discussed the DREAM NoC router architecture and hardware design focusing on the 2D mesh based topology till now. Now we present an overview of the multi-layered architecture incorporating the congestion aware adaptive routing described in Section 4.5. The 5x5 multi-layered NoC topology described in Section 4.5, is illustrated in Figure 6.13. In the Figure, tile numbering and addresses of each router in

$$(x, y)$$

format is included. We also discussed an extension of DREAM NoC to support this multi-layered NoC topology in Section 5.5. A brief overview of this multi-layered DREAM router architecture is also presented in this section.

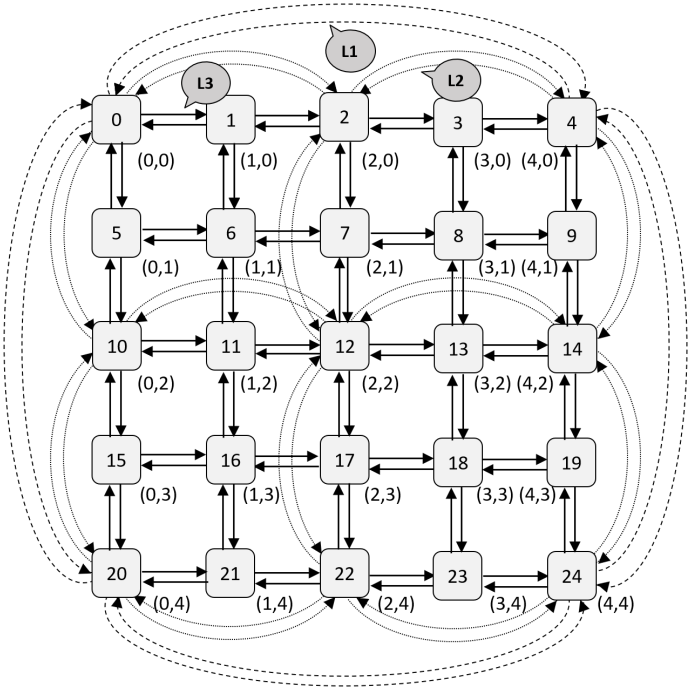


Figure 6.13: Detailed overview of the multi-layered NoC illustrating tile numbering assigned and addresses in (x,y) format

6.5.1 Multi-Layered NoC Router

An overview of the algorithms designed for multi-layered NoC is provided below and its detailed description is provided in Section 4.5. The following algorithms were integrated into the base NoC router.

- Static XY Routing (XY): Packets travel only on the 5x5 2D mesh (Layer3) using traditional XY routing
- Static Layer XY Routing (Layer XY): An extended version of the XY routing which utilizes all three layers. If source and destination is on the same network layer, regular XY routing is used to transmit packets on the

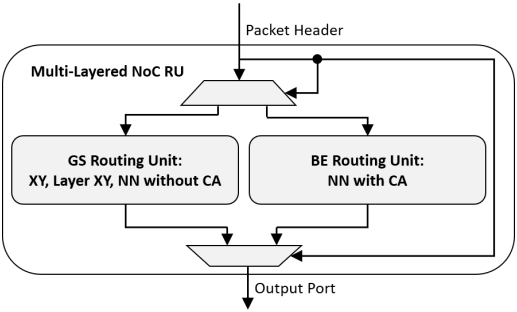


Figure 6.14: Routing unit for multi-layered NoC Router

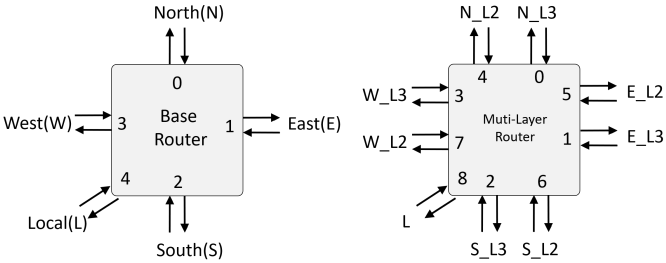


Figure 6.15: Port information connecting the different layeres in the multi-layered router. Base router shown for reference.

same layer. If the source and destination nodes are on different layer, the base 5x5 layer is utilized using XY routing to transmit packets.

- Static routing using Near Node Function without Congestion Avoidance (NN without CA): Packets are always transmitted on paths with the shortest hop count across the three network layers.

- Dynamic routing using Near Node Function using Congestion Avoidance (NN with CA): Packets are transmitted on shorter hop count paths during low traffic and routed across pre-determined longer hop counts during high traffic scenario.

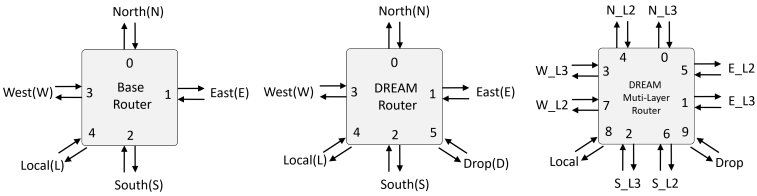


Figure 6.16: Number of ports for the different routers implemented. Base router for comparison, DREAM router for 2D mesh based topology and DREAM router for a multi-layered topology

The base router shown in Figure 6.1 was extended to support upto 9 links. As the number of ports range from 4 to 9 depending on the router's position in the network, unnecessary ports can be disabled. In the Figure 6.13, only ports connecting the neighbouring routers are illustrated. Local ports are not illustrated for simplicity. Deadlocks were avoided here, by utilizing virtual channels and restricting turns in the turn model at each layer. In our test implementation, the linkwidth is set to 32bit, this can be configured at design time. The routing algorithms were implemented in the routing unit and an overview is shown in Figure 6.14. Along with a SL filed in the packet header which indicates BE or GS packet, another special field is designed in the packet information section to indicate the type of routing algorithm to be used for a packet. A GS packet is routed according to the deterministic algorithms like XY, LayerXY or NN without CA based on this packet information. These algorithms route packets on the same paths and do not change at runtime. A BE packet is routed using adaptive NN with CA routing. This algorithm, routes packets along different paths based on local traffic information. With the increase in the number of ports, there is also increase in the input buffer and cross bar resources. One of the motivations to just connect alternate nodes in the lower levels of the network was to reduce the number of high port routers. A general overview of the ports connecting different layers for a multi-layered router is shown in Figure 6.15. A base router for a regular 2D mesh is also shown for reference.

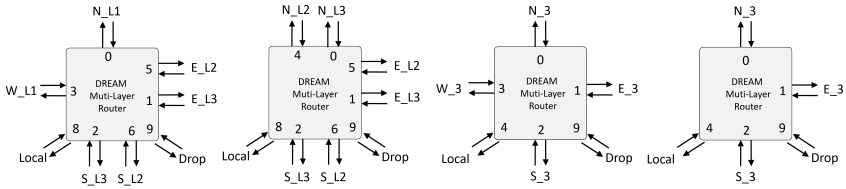


Figure 6.17: Examples of DREAM router for multi-layer network at different address locations (0,0),(0,2),(1,1) and (0,1). Refer Figure 6.13 for address locations

6.5.2 Multi-Layered DREAM NoC Router Extension

The DREAM router was extended to support the multi-layered NoC shown in Figure 6.13. The total number of ports for a regular 2D mesh topology is 6 including the Local (L) and Drop (D) port. For the multi-layer topology the maximum number of ports is 10, in our 5x5 test topology. A comparison of routers is shown in Figure 6.16. The number of ports varies depending on the position. It ranges from 5 to 10 and examples of the routers at different locations is illustrated in Figure 6.17. Deadlocks are avoided by restricting certain turns at each network layer to prevent cyclic dependencies. Thus such paths are not rewarded. To avoid livelocks, forward ant packets are dropped if the number of hops exceeds a certain threshold. In our design, for the 5x5 three layer multi-layer NoC used to investigate in this work, the limit is set to the diameter of the 5x5 mesh layer which is 8. This can be configured at design time.

6.6 Summary

In this chapter, we described the NoC architecture implemented and the supporting adaptive routing algorithms and multi-layered topology extensions. Section 6.1 provided an overview of the DREAM router architecture. This includes description of the base router used, DREAM routing units and packet formats. Section 6.2 provided design constraints and discussed how complex computations were handled to make the implementation hardware friendly. This also included

an overview of the modifications done to the original AntNet routing algorithm to simplify it with the aim of reducing hardware complexity. Functionality of the DREAM router was described in Section 6.3 which included how different packets like forward, backward and data packets are processed in the router. Deadlock and livelock avoidance techniques implemented is described in Section 6.4. Multi-layer extension and supporting routing algorithm was described in Section 6.5.

7 Results and Analysis

The architecture and evaluation of the contributions in this work like the DREAM NoC, supporting framework, multi-layered extensions and supporting adaptive routing algorithms is presented in this chapter. Its HDL implementation and simulation framework is described and the feasibility of the proposed techniques is presented. An HDL implementation which can be synthesized is developed to obtain resource requirement numbers and achievable clock frequency. An HDL implementation aids in FPGA-based prototype or an ASIC design. A proof-of-concept HDL implementation of the proposed NoC and supporting concepts is developed using SystemVerilog.

7.1 Evaluating Reinforcement Learning Based Routing

7.1.1 Overview of Test Environment

A test bench was developed to test the DREAM NoC and supporting features in SystemVerilog. An overview of the design is shown in Figure 7.1. It comprises of multiple test modules dedicated to each node in the network. As shown in the figure each individual test bench at each node injects and receives packets into the network. Each dedicated test bench comprises of a Network Interface (NI), which connects to the NoC. Here the NoC is an interconnection of routers according to a certain topology. In general, each router comprises of a Local port which is connected to its local processing tile comprising of data reception or generation

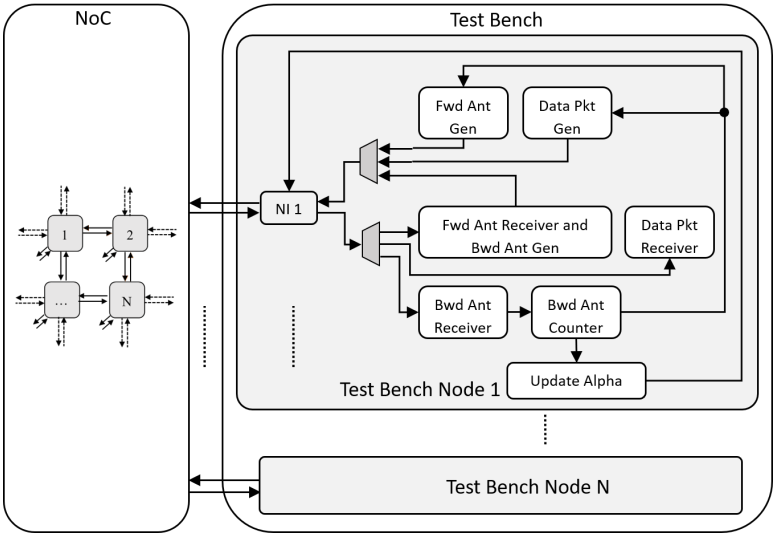


Figure 7.1: An overview of test bench

modules. Examples for such modules include, processing units or memory units. In the test setup, these processing tiles are represented by a dedicated test bench which injects data into the network according to a specific pattern and receives data. The data sent and received and monitored to compute metrics like data/ant packet injection rate, data/ant packet ejection rate (throughput), average delay etc.

Before data packets are injected, the special ant packets are injected to discover paths in the learning phase as explained in section 7.4 and illustrated in Figure 5.3 . When a learning phase is initiated, Forward Ant Generator at each node (Fwd Ant Gen module in Figure 7.1) injects forward ant packets into the network periodically according to the packet format in Figure 6.7. In the design, the forward and backward ant packets are of one flit size of 128 bits. If a forward ant packet successfully arrives at a destination router, it is converted to a backward ant packet by swapping source and destination fields in the router. The backward ant packet is according to the format in Figure 6.8. The backward ant packet is ejected by the router to the test bench via its local port. The test bench, using its monitoring units keeps track of the number of forward ant packets received

and converted and injects the backward ant packets back into the network. The backward ant packet then retraces its path back to the source node, updating the routing/pheromone tables in the routers on its way. It is then received by the test bench and a successful backward ant received indicates discovery of a feasible path between a pair of nodes. A counter is used to keep track of successful received backward ant packets which indicate discovery and/or update of paths information. In the learning phase, the focus is to discover at least one path between a pair of nodes while ensuring low learning phase overhead. Therefore, as soon a successful backward ant is received the forward ant packet generation for that pair of source and destination is paused to reduce unnecessary traffic. Once at least one backward ant packet is received for all pair of nodes, indicating at least one path is available for every pair of nodes in the network, the learning phase is ended. For network or a partition comprising of n nodes the learning phase ends when $n * (n-1)$ paths are discovered. The application phase is then initiated.

In the test module, to denote the application traffic, data packets are injected according to a Uniform Random Traffic pattern. The data packet generator module (Data Pkt Gen), injects data packets of varying flit length in a uniform random traffic pattern. Injection of packets according to a synthetic traffic pattern is widely used for performance evaluation. In uniform random traffic, each node in the network transmits packets to all other nodes of the network with the same rate. When the application phase is initiated, the forward ant packet generator resumes injection of forward ant packets at each node to discover better routes over time. As the special ant packets now travel along with the data packets, better paths can be now discovered based on application traffic patterns. Depending on if static or dynamic α value technique is chosen the α value is updated and transmitted to each router via the test bench accordingly. Monitoring units also keep track of data packets injected and received to compute throughput and latency.

7.2 Minimal and Non-Minimal Route Computation

We start with studying the capability of the ACO based routing in DREAM NoC to discover minimal and non-minimal paths in regular and irregular topologies. Regular or irregular partitions in a network can be created which haver regular or irregular topologies, either due to spatial isolation techniques or network component failure. Such topology agnostic routing schemes are beneficial to discovering routes within these partitions which are minimal and non-minimal. Minimal routing implies that the path between source and destination has the shortest distance referring to the Manhattan distance in a regular mesh based topology. XY routing scheme is a minimal routing strategy. Non-minimal routing is beneficial to distribute load in a network to avoid congestion. DREAM routing supports both minimal and non-minimal paths. Evaluations are conducted on regular and irregular topologies. Results presented include learning time overhead to discover paths and the learning time of topologies of varying degree of regularity is compared. The number of special ant packets needed to explore paths within partitions and the number of ants which are lost are analysed. The DREAM routing is tested on a range of partitions as shown in Figure 7.2. A 4x4 NoC is setup, and a partition under test is instantiated in the network. The border routers in this partition are directed to not transmit or receive packets outside of it. The results presented here include latency in cycles in terms of average hop count, learning time in cycles required to computes all paths in a partition, percentage of ant packets successful and overhead in terms of total ant packets needed.

The capability of the DREAM algorithm to find minimal and non-minimal paths in regular and irregular topologies is demonstrated using the results. Different partition shapes are chosen to test the algorithm and are organized into different categorizes as shown in Figure 7.2. In category 1, regions having regular mesh topology are considered with increasing number of nodes. The second category considers different irregular topologies interconnecting 8 nodes. The third considers three different topologies with 12 nodes. Topologies where non-minimal

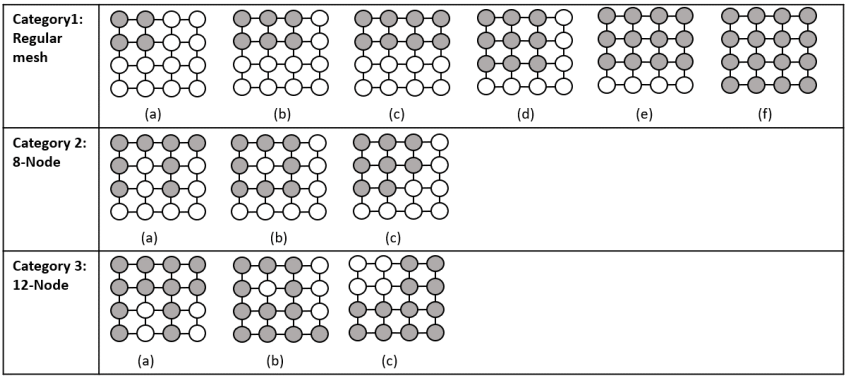


Figure 7.2: Range of partitions considered [21]

routing is required for successful communication between all pairs of nodes are also considered as shown in category 2a, 2b, 3a and 3b.

A summary of the techniques used to discover paths is presented here again. A forward ant packet which comprises of one flit is injected at each node in a region. Forward ant packets are periodically generated at each node. If the forward ant successfully reaches the destination, it returns back on the same path and updates the pheromone tables along the way thus distributing the routing information. A path is now said to be successfully discovered between a source and destination pair. If the forward ant is not successful it never returns to the source router as a backward ant packet. After a set waiting period elapses, the source node sends another forward ant packet. This is repeated until a path is discovered for a chosen destination node. Ant packets to that specific destination are not injected further after a path is discovered, to avoid adding unnecessary traffic in the network. The learning phase is considered complete here when every node has one discovered path to every other node in the partition.

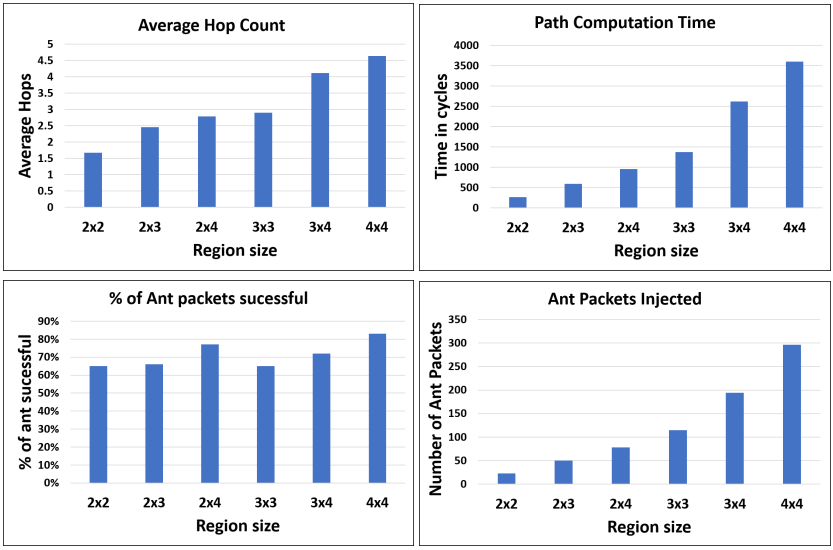


Figure 7.3: Results of partitions in category 1, mesh topologies [21]

7.2.1 Learning Time Overhead Analysis

Results for regular mesh topologies of category 1 is shown in Figure 7.3. The time required to compute paths (learning phase overhead) and the total number of ant packets injected increases with the increase in number of nodes. This is expected as the number of nodes increase, the number of forward ants injected also increases. The success rate of the forward ant packets range from 65% to 83%. Let us elaborate this using an example of a 2x2 region size. Total forward ant packets injected by the 4 nodes to discover at least one path between all pairs is 23. Among them 15 (65%) forward ants successfully reached its destination, and translated to backwards ants. In mesh topologies, the nodes are clustered together, and have high forward ant success rate when compared to irregular topologies as we can see next.

Let us now compare results between regions having the same number of nodes but different topologies. Challenging topologies which having high degree of

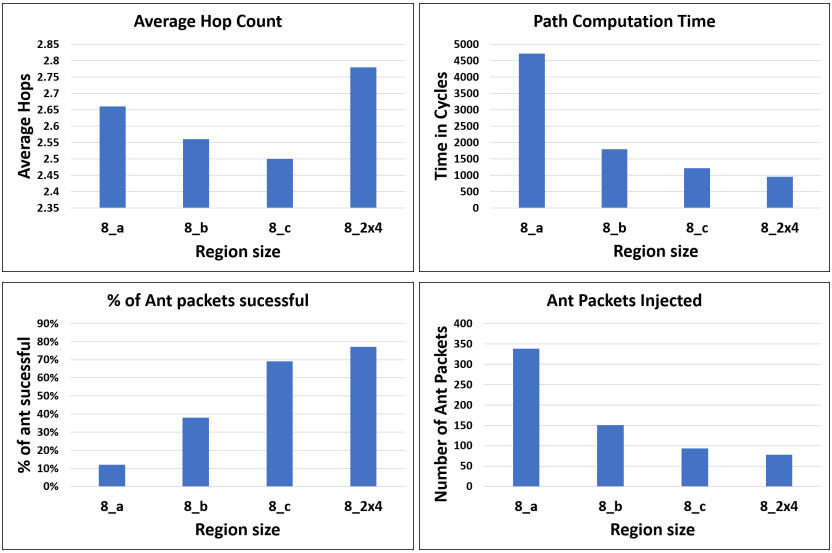


Figure 7.4: Results of partitions in category 2, 8-node topologies [21]

irregularity leading to non-minimal paths is considered as shown in Figure 7.2. Results for partitions in category 2 are shown in Figure 7.4 and compared against mesh with the same number of nodes. The learning time overhead is the highest for the first partition (a) which has a high degree of irregularity since more number of non-minimal routes need to be computed. The percentage of success is the lowest for this partition because it is the least clustered, thus available feasible paths in all directions is limited. Therefore more packets need to be injected to discover feasible paths. Similar results can be seen for category 3 as shown in Figure 7.5 and compared against mesh with the same number of nodes The total number of packets injected is presented next to analyse the packet injection overhead. For partitions in category 3, which has 12 nodes in a region, it required around 200 packets for a mesh based region to discover at least one path between all source destination pairs as shown in Figure 7.5. The time taken was around 2600 cycles. This increased to 450 packets when the regularity of the topology decreased and non-minimal paths were required. The learning time overhead increased upto 8200 cycles.

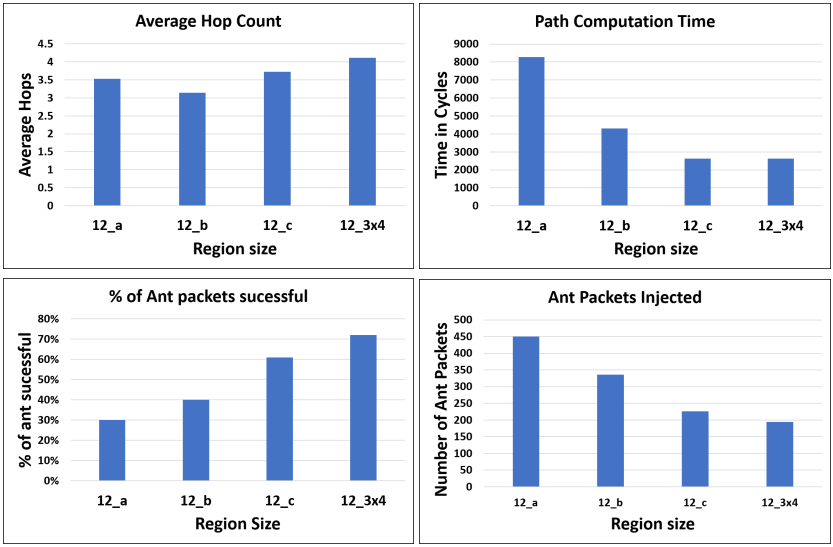


Figure 7.5: Results of partitions in category 3, 12 node topologies [21]

Table based routing has an inherent resource overhead when scaled up due to the implementation of the tables. For larger network sizes, if the maximum possible number of nodes in a partition is known, then the size of the routing/pheromone tables can be fixed accordingly so as to avoid having all the source destination pair information stored in the table. Preliminary hardware utilization for a single regular 5-port XY router and an ACO router is shown in Table 7.1. The synthesis targets Virtex-7 (XC7V2000T). The regular XY router here is the base router architecture, using 32 bit link width and implementing XY routing. Since the bit width chosen for ACO is 128 bits to have a single flit ant, this resulted in the input buffers consuming most of the resources in ACO router. AntMem

Table 7.1: Resource usage [21]

XY	total	buffer
LUT	2284	1479
FF	4546	4184

AntNoC	total	buffer	AntMem
LUT	7879	3461	642
FF	6586	4196	476

module is where the pheromone table is designed, which has the probability of attractiveness of each port when targeting a certain destination. To reduce the resource consumption, an alternative is utilizing ant packets having multiple flits and thus reducing the link width and input buffers. Overall the ACO based router, while highly flexible, does consume more resources and further optimizations are possible. This flexibility can be beneficial for large scale platforms, where the alternative of utilizing an ad-hoc routing algorithm for each new partition shape can incur high software overhead. The results presented in this section was published in [109].

7.3 Runtime Adaptive Alpha Technique

Next we investigate the ACO based DREAM routing scheme further focusing on exploiting the trade-off between exploration and exploitation which is represented by the α factor in Equation 4.4. The aim here is to study how one can decrease the learning time overhead by adapting the α over time. The results presented in this section was published in [109]. Learning phase is used to discovering preliminary paths in a given partition or network . After this period, the application packets are injected and special ant packets travel in parallel to determine optimal routes over time. It is desirable for this learning time period to be low. How different α values for the learning phase and application phase can be used to improve overall performance was described in Section 5.3. Here we present the results and study the impact of the α factor on this time period and the effect on the degree of regularity of the network on the time required.

According to the AntNet algorithm, best value of α varies between [0.2, 0.5], beyond which there is oscillations in performance. We test α values within this range and investigate lower values of α like 0 and 0.1 as well Such low values of α do not take into account local traffic fluctuations. In the learning phase, only special ant packets travel in the network and thus traffic fluctuations are low. We conduct tests using static and dynamic α technique shown in Figure 5.6 on selected regular and irregular topologies which are shown in Figure 7.6. We obtain the

results of learning time overhead, latency and throughput. The injection rate is set to when the throughput starts to saturate for each of the topologies.

Initially static α tests are conducted where the α value chosen at design time remains constant throughout the learning time and application runtime. Different values of α are tested for the selected topologies. The α values chosen here are in the range between 0 and 0.5. For a given topology, these static tests are conducted to investigate, which α value yields low learning time and which α value results in high throughput. The α value which achieves the lowest learning time is selected for $\alpha 1$ and the α value which achieves a high throughput and low latency is selected for $\alpha 2$ for dynamic alpha technique shown in Figure 5.6.

Later we test the dynamic α technique on the selected topologies. The $\alpha 1$ and $\alpha 2$ values obtained using static tests are utilized when evaluating the dynamic α technique. During the learning time $\alpha 1$ is used. Once the learning time is completed, the test bench dynamically updates to $\alpha 2$ in all the routers of the network. Each successful backward ant packet received indicates a discovery of a feasible path between a pair of nodes. When at least one path is successfully discovered between all pairs of nodes, this triggers the end of learning time. Then the application phase is initiated and the data packet generator in the test bench starts injecting packets into the NoC. Special ant packets continue to be generated once again to find optimal routes and travel along with the data packets. This results in the initial pheromone table built inside every router is based on $\alpha 1$, which resulted in low learning time in static tests. And during application run time, $\alpha 1$ is changed to $\alpha 2$, and the pheromone tables are now updated based on $\alpha 2$ which yielded high throughput in the static tests. The results presented here are generated using HDL simulator Modelsim.

7.3.1 Improving Learning Time Overhead

Results for topology (a) which is a 3x3 mesh network as shown in Figure 7.6, are shown in Figure 7.7. The α of 0.35 yields a lower learning time but α of 0.25 results in better throughput. When combined in the dynamic alpha technique there

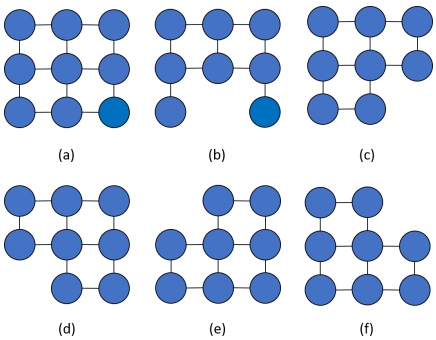


Figure 7.6: Regular and irregular topologies under test for static and dynamic α technique [109]

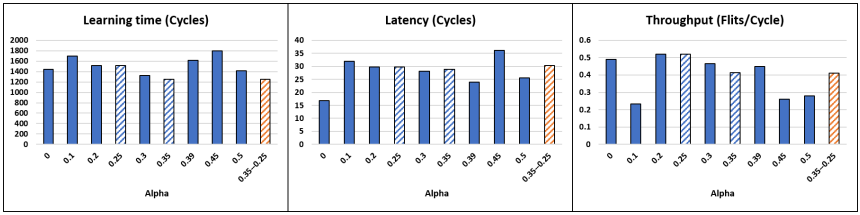


Figure 7.7: Comparison between different cases of static α and dynamic α (α1=0.35 to α2=0.25). Results for topology (a) [109]

is a slight improvement in learning time, with comparable latency but decrease in throughput. Results for topology (b) in Figure 7.6, are illustrated in Figure 7.8. In this topology, α of 0.20 was selected for α1 even though α of 0.4 resulted in a slightly lower learning time. This was done as α of 0.20 had a much higher throughput compared to α of 0.4. The α2 was set to 0.45 as this resulted in the highest throughput. Results for topology (c) in Figure 7.6 are illustrated in Figure 7.9. The α value of 0 here results in the lowest learning time. But the α value of 0.4 results in the highest throughput and low latency. Here we set α1 as 0 and α2 as 40. This results in the best performance for both, where the learning time is now decreased by 51% with comparable latency and throughput when compared to static α technique of 0.4 .

Similar tests are conducted across the other three topologies (d), (e) and (f) in Figure 7.6. These topologies are similar to (c) as one corner node is removed

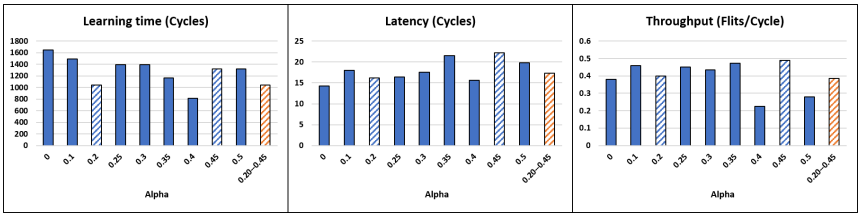


Figure 7.8: Comparison between different cases of static α and dynamic α (α1=0.20 to α2=0.45). Results for topology (b) [109]

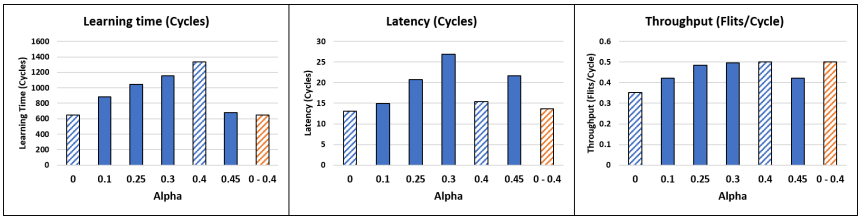


Figure 7.9: Comparison between different cases of static α and dynamic α (α1=0 to α2=0.4). Results for topology (c) [109]

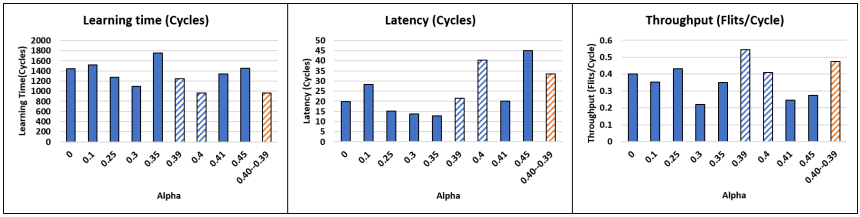


Figure 7.10: Comparison between different cases of static α and dynamic α (α1=0.40 to α2=0.39). Results for topology (d) [109]

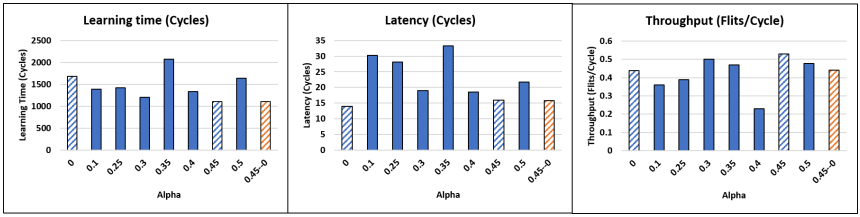


Figure 7.11: Comparison between different cases of static α and dynamic α (α1=0.45 to α2=0). Results for topology (e) [109]

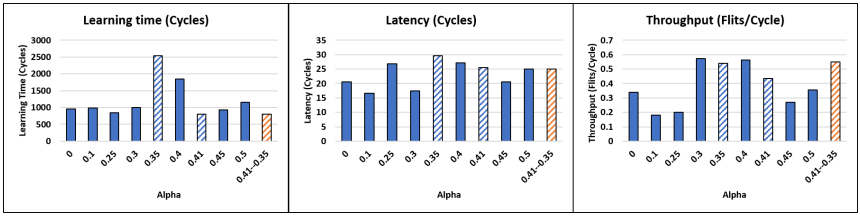


Figure 7.12: Comparison between different cases of static α and dynamic α (α1=0.41 to α2=0.35). Results for topology (f) [109]

in all of them and orientation has changed. These tests were still conducted to investigate if the initial pheromone table values present in each router has an effect on performance. At design time, pheromone table values were set and the values were not set to zero. Since the topologies under test were 2D mesh based network, initial pheromone table values were based on the concept of X-Y routing. Initially, the forward ant packet start travelling on paths using pheromone values based on the X-Y routing concept and over time the pheromone table values are updated accordingly.

Results for topology (d) are shown in Figure 7.10. Here we test the network using varied values of α to analyse the learning time better. After running the static tests, α1 and α2 was set to 0.4 and 0.39 respectively. The resulting latency and throughput of the dynamic test was better than when α was 0.4 with the same low learning time overhead.

Results for topology (e) are shown in Figure 7.11. Here we see that α of 0.45 resulted in the lowest learning time, latency and throughput. Since latency of when

α was 0 was marginally better than when α was 0.45, we applied the dynamic α technique where $\alpha 1$ and $\alpha 2$ are 0.45 and 0 respectively. Here we see that static α of 0.45 performed better than the dynamic α. AntNet[36] algorithm describes that when α is close to 0, fluctuations of performance can occur because it is unable to follow sudden variation in traffic pattern. Here we see that, in this case adapting it to 0 had a slight increase in latency and decrease in throughput.

Results for topology (f) are shown in Figure 7.12. Here we see that multiple values of α results in low learning time. We select $\alpha 1$ to be 0.41 since its latency and throughput is better compared to other values. We update $\alpha 2$ to 0.35, as it has a higher throughput. Here, the dynamic α technique shows a decrease of 68% in learning time compared to static α of 0.35. Compared to static α of 0.35, there is also a decrease of 15.4% in latency and the throughput is the same. The adaptive α technique reflects the low learning time of static α of 0.41 and throughput performance of static α of 0.35.

The ACO based DREAM routing algorithm explores networks in partitions for feasible paths based on history of already utilized paths and local traffic information. We investigated dynamic setting of the α which represents the trade off between exploration vs exploitation of already discovered paths to decrease the overhead of learning time. With the aim to decrease learning time overhead, by proposing an adaptive exploration technique, a decrease in the learning time overhead by upto 68% could be observed. The proposed dynamic α technique yielded better results for certain topologies when compared to others.

7.4 DREAM NoC and Supporting Framework

The DREAM NoC and supporting framework are designed using SystemVerilog and evaluations are done using HDL Modelsim simulator. The results presented in this section was published in [13]. Here we evaluate the learning phase and application phase period. We present results of latency and throughput of the data traffic which are routed based on the routing tables built at runtime by the

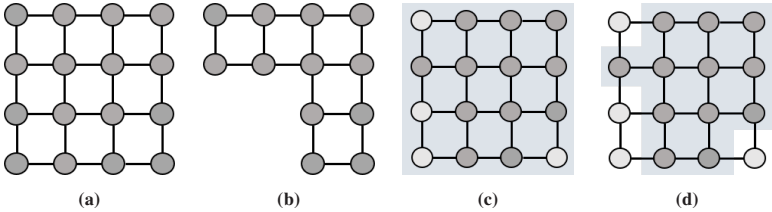


Figure 7.13: Test cases for evaluation [13]

DREAM routing mechanism. Based on the results in section different α values are used for learning and application phase to improve overall performance. They are represented as $(\alpha 1, \alpha 2)$ in our evaluations for learning and application phase respectively. We compare latency and throughput DREAM routing algorithm to other topology agnostic algorithms like UP*/Down* (UD) [84] and Logic Based Distributed Routing (LBDR) [88]. We also test against traditional XY routing on a regular 4x4 mesh for reference. We implemented UD and LBDR routing techniques based on [84] and [88] respectively. For UD, in our tests, routing table information was computed at design time for all the topologies under test. The routing table in each router was set at design time. For LBDR, the routing restrictions in our evaluations are based on the UD routing as described in [88] and the special bits to define regions and routing are set at design time. We implemented both UD and LBDR in SystemVerilog and integrated it with the base router architecture for fair comparison.

7.4.1 Latency and Throughput Analysis

We start the evaluations on a regular 4x4 mesh in Figure 7.13a. During the application phase, synthetic uniform random traffic is injected at each node by the test bench. Here $(\alpha 1, \alpha 2)$ is set to (60, 20). Throughput and latency results are shown in Figure 7.14. The throughput of DREAM routing is comparative to XY and it performs better than UD and LBDR which are similar.

The latency of ACO based DREAM routing has a *remarkably* different S-curve compared to the other algorithms. At low injection rates, it has the higher latency

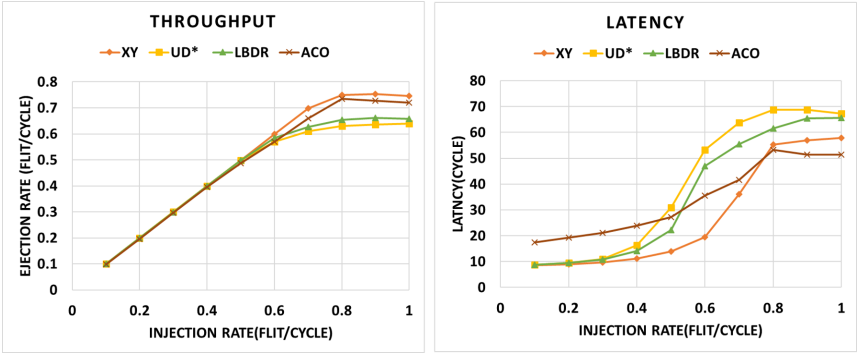


Figure 7.14: Results for a regular 4x4 topology as shown in Figure 7.13a [13]

and at high injection rates, it has the lower latency. At saturation, compared to UD we see upto 22% decrease in latency. The reason for this difference in the S-curve is because ACO based routing searches for and routes packets on non-optimal paths as well. At low traffic scenarios, longer hops are explored and used. Minimal and non-minimal paths are explored even at low traffic. Only minimal paths are not used and over time when the traffic increases, there is a jump and it stabilizes around 0.8. This is one of the advantages of the ACO based routing: routing across non-optimal paths result in discovering new routes and distributes traffic across the network. The injection rate of ant packets is based on Eq.5.5, c is set to 1 here. So injection of the forward ant packets is at the same rate as data packets. At low data injection rates, the number of forward ants are fewer as well. Over time with more long term traffic information better paths are discovered and performance improves as shown in the Figure 7.14. This is due to the reinforcement learning nature of the algorithm.

Next we test the algorithm on an irregular topology as shown in Figure 7.13b and the results are shown in Figure 7.15. We compare against UD and LBDR as XY does not support such a topology. Results when different α values are set for learning and application phase is also evaluated and shown in the figure. The ant packets here are injected at twice the injection rate. There is a 25% increase in throughput compared to other algorithms with the α pair $(\alpha 1, \alpha 2)$=(20,20). The latency S-curves of the ACO based routing are *remarkably* different once

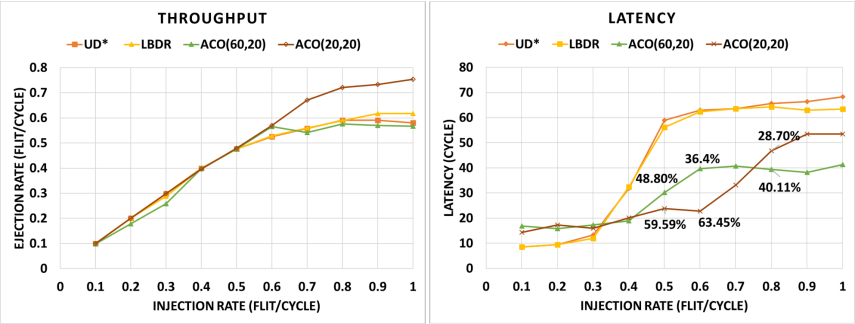


Figure 7.15: Results for an irregular topology as shown in Fig. 7.13b [13]

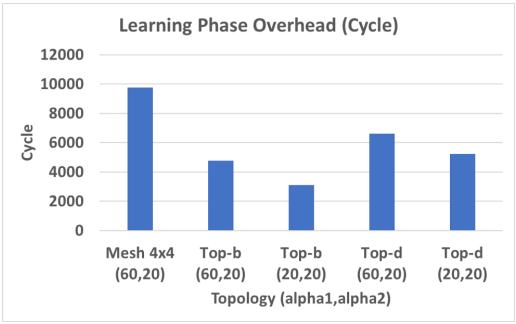


Figure 7.16: Learning phase overhead[13]

again when compared to the other algorithms. The latency jump occurs at higher injection rates. The latency starts to saturate approximately for UD (and LBDR), ACO (60,20), ACO (20,20) at 0.5, 0.6 and 0.8 injection rates respectively. The decrease in latency at these points is also shown in the Fig. 7.15. ACO(60,20) has a decrease of 48.8%, 36.4% and 40.11% at 0.5, 0.6 and 0.8 respectively. ACO(20,20) has a decrease of 59.29%, 63.45% and 28.70% at 0.5, 0.6 and 0.8 respectively. With the increase in injection of the ant packets, optimal routes are discovered over time as paths are computed using a combination of long term and local traffic fluctuations. The performance improves over time as shown in the Figure 7.15. Since the paths are not static like UD, alternate paths are also dynamically chosen at runtime. Learning time overhead is shown in Figure 7.16.

7.4.2 Application Test Case

Now we test the DREAM NoC with traffic generated by the testbench based on a Communication Task Graph (CTG) of an MPEG4 video decoder. The CTG in MBps and its mapping on a 4x4 mesh is based on [114] and is shown in Figure7.17. The maximum achievable frequency for the proposed DREAM router with the DREAM routing algorithm is approximately 107MHz for a three stage pipeline. This was evaluated by synthesizing for a Virtex-7 FPGA (xc7vx485tffg1761-2). The router can be further optimized by increasing the pipeline stages to support higher frequencies. For our text case we set the frequency to be 100MHz. The CTG in MBps is converted to flits per cycle and using this value the simulator test bench injects the required traffic from each node. Packet sizes vary from 1 to 15 flits. The results are shown in Figure 7.18.

The data packets injected comprises of a header, body and tail flits and data flits vary from size 1 to 15. MPEG4 application is tested on topologies illustrated in Figure 7.13c and Figure 7.13d. Figure 7.13c, is a regular partition, where the topology is a 4x4 and only the nodes running the MPEG4 application inject traffic. Figure7.13d, is an irregular partition. Throughput results are comparable to XY and UD on a regular partition but latency is slightly higher. On an irregular partition, XY routing is not supported, but throughput results are still comparable and the latency is slightly higher. This is because the overall ant injection rate was kept low to maintain throughput.

Comparison of single router resource utilization targeting the Virtex-7 FPGA between the ACO based DREAM router and the router running UD routing algorithm is shown in Table 7.2. ACO consumes more resources than UD due to the routing computations involved and more input buffer resources. This is because, an extra port was added to the ACO router to drop the ant packets which were not successful. We see that more than half of the router resources are consumed by the buffers and further optimizations are possible by improving this feature. In this section, we evaluate the DREAM NoC along with its supporting framework to aid in flexible spatial isolation for mixed-critical platforms.

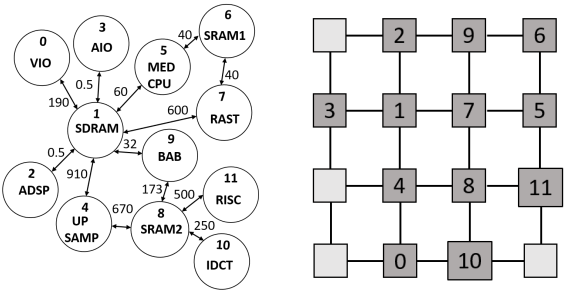


Figure 7.17: CTG of MPEG4 video decoder and mapping [114] [13]

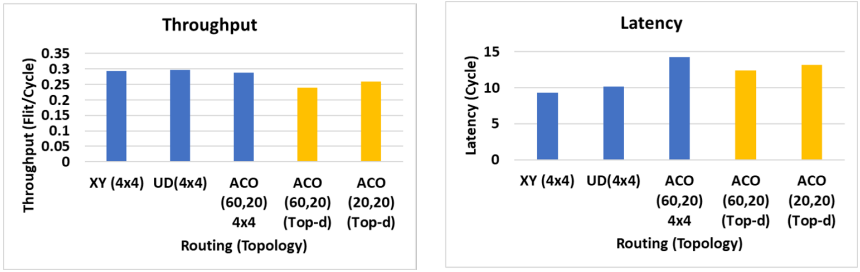


Figure 7.18: Results for MPEG4 video decoder [13]

7.5 Evaluating Adaptive Block-Based Multicast Routing

At the routing level we also proposed to improve flexibility by proposing Block-Based multicast routing as described in section 4.4 We implemented the block-based multicast routing on a Virtex7 FPGA VC707 board and tested it using traffic patterns seen in a fault tolerant application implementing redundancy. A brief motivation for using this test case is described next.

Table 7.2: Resource utilization comparison of a single router[13]

UD	Total	Buffer
LUT	4054	2151
FF	4060	2780

ACO	Total	Buffer
LUT	5739	2886
FF	4985	2780

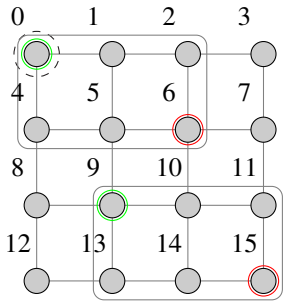


Figure 7.19: A 4x4 Mesh with multicast groups MG1 and MG2 [100]

7.5.1 Fault Tolerant Application Use Case

Functional and data flow-based programming models are well known in the field of high-performance computing (HPC). Such models target efficient programming of highly parallelizable and distributed software. The communication in such software is typically done via MPI. MPI is a standardized collection of functions for communication and data transfers in distributed-memory and shared-memory computing systems. Besides the Point-to-Point operations send and receive, MPI defines a broadcast operation, where data is distributed to multiple processes. An efficient hardware support for broadcast operations leads to a significantly lower communication overhead for MPI and dataflow driven software applications.

Fault-tolerant applications benefit from broad-and multicast hardware support. Fault tolerance is typically implemented by redundancy and comparison of redundant results. Realizing redundancy is done by executing the same operation multiple times, either sequentially in time or parallel in space. Double or triple modular redundancy (DMR and TMR) uses redundant hardware components and applies a voter to detect faults in the results. In a NoC-based system the redundant

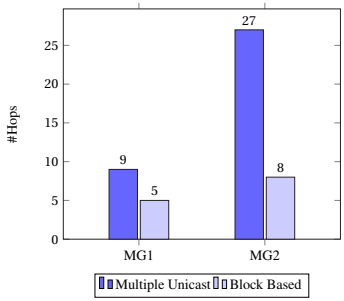


Figure 7.20: Comparison of hops for the two multicast groups shown in Figure 7.19 [100]

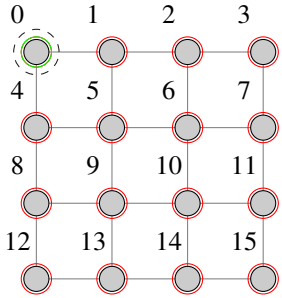


Figure 7.21: A 4x4 Mesh with node 0 as src and DstS. DstE varies from node 1 to node 15 [100]

components are mapped on separate processing units in the NoC. Broadcasting or multicasting instead of individual transmission of the input data achieves a speed up and reduces the load of the NoC.

We implemented a 4x4 NoC as shown in Figure 7.19 running on a clock frequency of 50MHz. For sake of readability we refer to nodes using IDs shown in the figure. Redundant tasks are mapped onto multiple nodes in two regions which are highlighted. Memory tile is located in node 0 and data needs to be read from the memory and transmitted from node 0 to nodes 1,2,4,5,6 forming one multicast group (MG1). Node 0 needs to send another set of input data to nodes 9,10,11,13,14,15 forming another multicast group (MG2). Using the block-based multicast, (DstS,DstE) for MG1 and MG2 are (0,6) and (9,15) respectively.

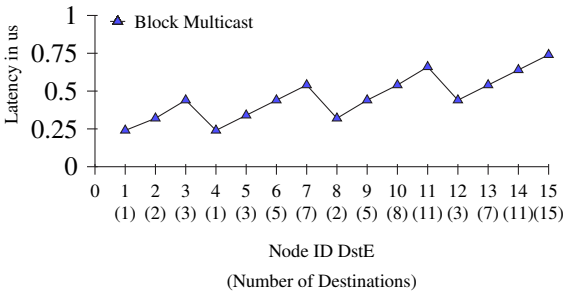


Figure 7.22: Latency of packets with Block-Based multicast when DstE varies from 1 to 15 [100]

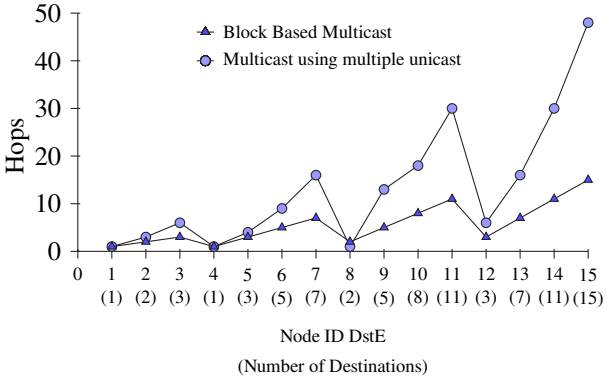


Figure 7.23: Comparison of hops [100]

7.5.2 Performance

Results for total number of hops for the test case in Figure 7.19 is shown in Figure 7.20. It can be seen that paths taken by packets when using block-based multicast has a lower hop count than paths by packets using multiple unicasts which uses DoR. The performance is better when the destination group is far from the source node as can be seen from the results of MG2. This is due to using tree based routing only within the block which restricts traffic to within the block and uses common paths as far as possible. Address Overhead (AO) is also lower, when using block-based the AO is 10 (16 with BV) and when using

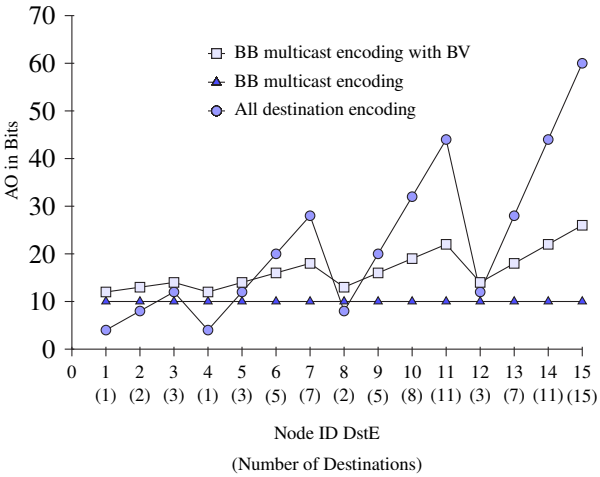


Figure 7.24: Comparison of Address Overhead [100]

multicast with all destination encoding AO is 24 bits. A 33% decrease in AO is seen even with BV. Let us consider a more general case where source and DstS is node 0 and DstE varies from node 1 to 15 as shown in Figure 7.21. In this case we consider all the nodes within the block whose size is varying as destinations. The variation in latency for block-based multicast is shown in Figure 7.22. Comparison in terms of hop count when multiple unicast messages are used is shown in Figure. 7.23. Block-based multicast performs better since it uses common paths as far as possible.

7.5.3 Address Overhead and Resource Utilization

Consider the general case shown in Figure 7.21. Each node address is represented by 4 bits. Two bits each for X and Y coordinate. The AO for this case is shown in Figure 7.24. We can see that even with BV encoding the proposed technique scales well when compared to multicast which uses all destination encoding used in most multicast techniques. Now let us consider a case when the size of the network increases. In larger mesh networks the address size gradually increases,

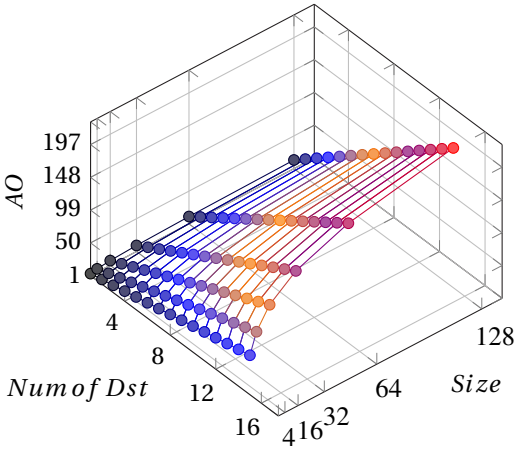


Figure 7.25: AO of multicast when all destination encoding is used [100]

which contributes to greater AO in multicast packets. To illustrate the benefit of the Block-Based technique we consider 16 potential destinations grouped together in a block of 4x4 in a 2D Mesh network. We look at networks of sizes 4x4, 8x8, 16x16, 32x32, 64x64 and 128x128. In Figure 7.25 AO of multicast packets using all destination encoding is presented, and results of block-based is shown in Figure 7.26. AO of block-based scales better and has a 79% decrease in overhead for 128x128 NoC. Resource utilization of a single router implementing Block-Based multicast is shown in Table. 7.3.

Table 7.3: Resource Utilization [100]

Resource	BB Router	BB Unit	% Overhead
Slice LUTs	5341	388	7.2%
Slice Reg	3734	49	1.3%
Slices	2035	385	23%

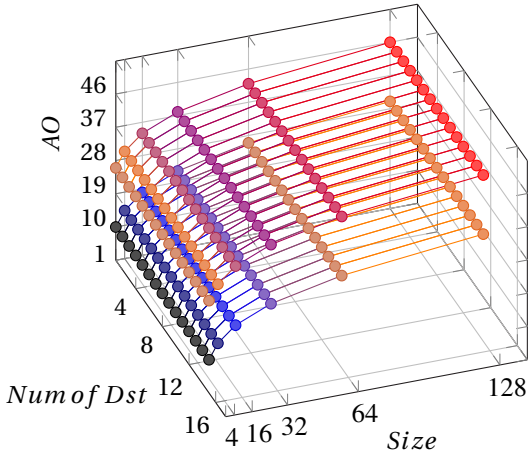


Figure 7.26: AO of block-based multicast with (above graph) and without (below graph) bit vector [100]

7.6 Evaluating Multi-Layered NoCs

In Section 4.5, we described developing multi-layered NoCs with adaptive congestion aware routing targeting mixed-criticality systems. In this section, we present the evaluations of such a network and the results were published in [105]. The routers are designed using SystemVerilog and results are generated using HDL simulator Modelsim. We present the results for the 5x5 three layer NoC shown in Figure 4.15. The results of the proposed congestion aware algorithm are compared against the popular XY routing, a modified XY routing for layered mesh and the static algorithm without congestion described in Section 4.5. Modified version of the XY routing was developed here to compare against the proposed routing against. It functions as follows. If the source and destination nodes are on the same layer then XY routing is used to transport the packet using the routers on same layer. For example if the source and destination nodes are on a Layer 2, then a packet is transported only on routers on Layer 2. If the source and destination belong to different layers, regular XY routing is used utilizing only routers on the base layer which is Layer 3.

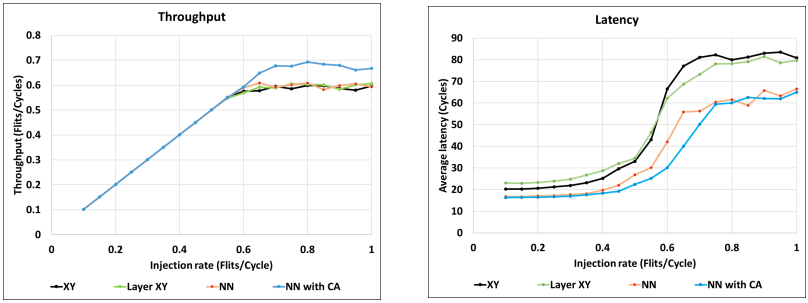


Figure 7.27: Throughput and Latency in cycles[105]

Data is injected into the network based on an uniform random traffic to test the algorithms. In this traffic case, each node has equal probability to send packets to every other node. Thus all pairs of source destination pairs are considered here. The packet length is also varied to show the influence on congestion and how the proposed algorithm with congestion avoidance performs better with increasing packet length. We use an FPGA platform to obtain resource utilization for the approach. A summary of the algorithms evaluated and compared is shown below:

- XY Routing(XY): Traditional XY Routing utilizing only 2D mesh (only the base 5x5 layer)
- Layered XY routing(Layer XY): Modified XY routing utilizing all the three layers
- Near Node Function without Congestion Avoidance (NN without CA)
- Near Node Function using Congestion Avoidance (NN with CA)

Throughput and latency results when the packet length is 10 are shown in Figure 7.27. With the integration of congestion avoidance there is an improvement of upto 16% in throughput. For latency, the algorithm with congestion avoidance has a decrease of upto 56% in latency compared to XY routing and decrease of upto 29% in latency when compared against the algorithm without congestion avoidance. At this packet length, the throughput of XY, Layer XY and NN are

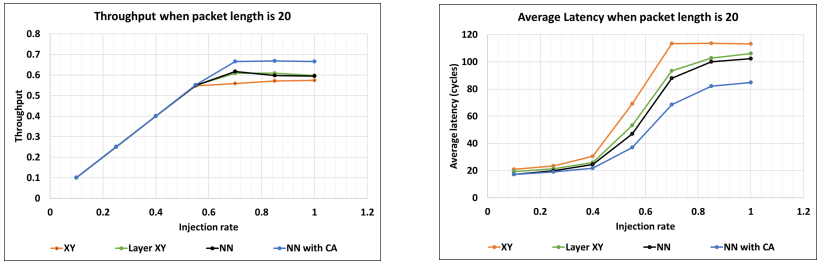


Figure 7.28: Comparison of throughput and latency for packet length 20[105]

comparable. The NN algorithm, always transports packets on paths with the shortest hop count. This can lead to creation of hotspots in the network which is eased by directing packets on longer hop paths when available. This results in increase in throughput of NN with CA algorithm when compared with the other algorithms with the increase in injection rate. With the increase in packet length, we can see in Figure 7.28 the algorithm without CA increasingly performs better than traditional XY routing. There is an improvement in the results of the algorithm with congestion avoidance when compared against the others.

The effect of packet length on latency is presented in Figure 7.29 by comparing performance under different packet lengths. With the increase in the injection rate, it can be seen that the saturation value for latency is lesser for NN with CA routing. For packets of length 5 for example, the improvement is less when compared against longer packet lengths of 10 and 20 as is shown. With the presence of longer packets, number of occupied routers increases for a single packet thereby increasing the possibility of hotspot especially when there is a high injection rate. Using congestion avoidance the packets are more evenly distributed across layers. The packets are forwarded on less frequently used paths (long hop) thereby decreasing overall latency.

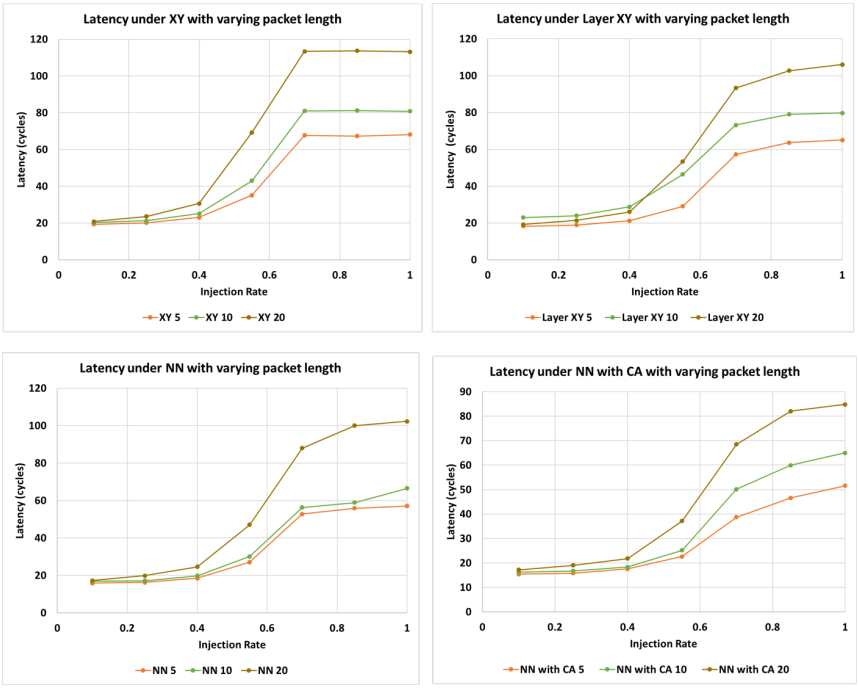


Figure 7.29: Comparison of latency under varying packet lengths[105]

7.6.1 Resource Overhead

An important drawback in such a multi-layered NoC implementation is resource utilization. Resource utilization of a single routing unit is presented in Table 7.4 for the different algorithms. Target Platform is Virtex-7 FPGA (7vx485tffg1761-2) and tool used is Xilinx Vivado 18.3. Resources utilized by the routing unit for different algorithms is shown and it can be seen that the proposed routing algorithm uses more resources, which is expected when compared with XY routing. Such a routing unit is present at each port of the router. Even with increase in complexity of the routing algorithm and the number of ports, the resources utilized by the routing units in the router is considerably lesser when compared against the rest

Table 7.4: Resource utilization of one routing unit. For comparison, percentage of resource consumed by one routing unit in a 5-port router is also shown [105]

Resource	XY (%)	LayerXY (%)	NN (%)	NN with CA (%)
Slice LUTs	6 (0.094%)	6 (0.094%)	93 (1.46%)	305 (4.8%)
Slice Reg	0	0	4 (0.1%)	4 (0.1%)

of the router resources as shown in Figure 7.30. The input buffers in the router here occupy a major portion of the resources.

Multi-layered networks can be beneficial for large scale networks. For small scale networks and in situations where severe resources constraints are present, the the proposed multi-layered network is not profitable due to its high port count routers. Such multi-layer routers occupy more resources than a router targeting a regular 2D mesh topology. But with increasing number of tiles being utilized in SoC platforms, the network size growing, the presence of layers and adaptive algorithms can yield benefits as can be seen in terms of latency and throughput here. At the expense of fraction of more resources, the improvement seen in latency and throughput is profitable in such circumstances.

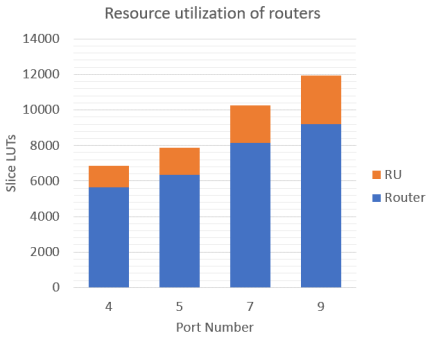


Figure 7.30: Comparison of router resource utilization [105]

In this section we evaluated a multi-layered hierarchical NoC targeting mixed criticality systems with an adaptive congestion aware routing algorithm. The

NoC comprised of multiple layers which the adaptive routing algorithm utilized to evenly distribute the packets depending on the availability of the links and criticality of the packets. Since multiple applications of varying criticality are sharing the same communication resources like routers and links, critical applications are prioritized and assigned the shortest hop paths. In high traffic scenarios, non-critical packets are assigned paths with longer hop counts to decrease influence over critical packets. Thus decreasing the congestion scenarios for critical applications and improving overall performance.

7.7 Multi-Layered DREAM NoC

The DREAM routing algorithm was integrated into multi-layered NoC described in the previous section and the evaluations are presented in this section. The evaluations presented here is using HDL simulator Questa Sim-64 2019.4. The testbench described in 7.1.1 is used along with the DREAM framework and the special ant packets and the data packets under uniform random traffic patterns is injected accordingly. Here the parition under test is the entire 5x5, three layer topology. Alpha value in equation 4.4 is set to 0.2 based on [36], where the authors describe that alpha value between 0.2 and 0.5 has less oscillations in performance

The proposed ACO based DREAM routing is compared against other static and adaptive routing algorithms targeting multi-layered NoCs described in section 7.6. An overview of the algorithms is provided below. The following algorithms are implemented in SystemVerilog and integrated into the same base NoC router for fair comparison

- Static XY Routing (XY)
- Static Layer XY Routing (Layer XY)
- Static routing using Near Node Function without Congestion Avoidance (NN without CA): Packets are always transmitted on paths with the shortest hop count across the three network layers.

- Dynamic routing using Near Node Function using Congestion Avoidance (NN with CA): Packets are transmitted on shorter hop count paths during low traffic and routed across pre-determined longer hop counts during high traffic scenario.

7.7.1 Latency and Throughput Analysis

Throughput and latency of the DREAM routing for the multi-layer topology is shown in Figure 7.31 and Figure 7.32 respectively. Packet length is set to 10. Dynamic routing algorithm have better throughput when compared to static routing. The proposed ACO based DREAM routing has upto 42% increase in throughput when compared to static XY routing and upto 29% increase in throughout compared to the adaptive congestion avoidance routing algorithm. There is improvement of upto 75% in latency when compared against static XY routing and upto 65% decrease in latency when compared to the adaptive congestion avoidance routing.

The adaptive routing algorithm used for comparison, takes into consideration the buffer level of only the downstream router when choosing the output port at each router. The proposed reinforcement enabled DREAM routing algorithm computes path based on local network information and history of travelled packet information. The forward ant packet when discovering new paths between nodes, takes into consideration buffer levels of all the router levels in its path. This is represented by the link attractiveness in equation 4.4. Due to the distributed learning feature of the algorithm, better paths are discovered. Variation in latency with different packet lengths is shown in Figure 7.33. Longer packets occupy multiple routers and can result in higher overall latency with increase in average packet length.

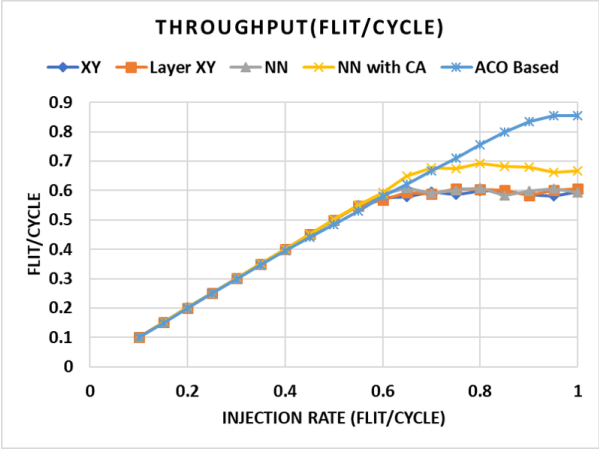


Figure 7.31: Throughput results for a 5x5, three layered topology as shown in Figure 4.15 [106]

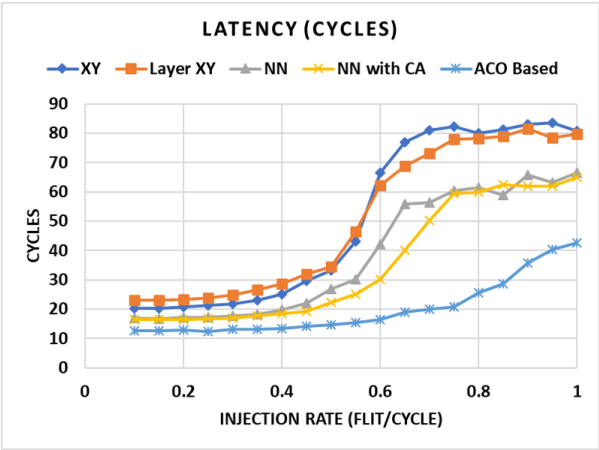


Figure 7.32: Latency results for a 5x5, three layered topology as shown in Figure 4.15 [106]

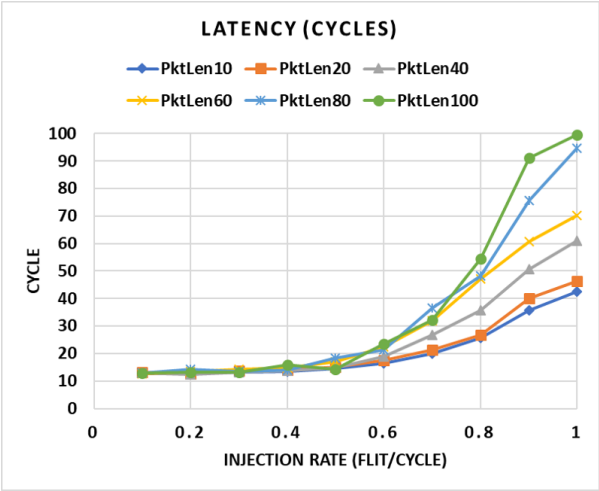


Figure 7.33: Latency results for ACO based routing. Testing for different packet lengths for a 5x5, three layered topology as shown in Figure 4.15 [106]

7.7.2 Learning Time and Hardware Overhead

An overhead is the learning time period required to build preliminary routing tables before the application can use the network. For the topology under test which is a 5x5, three layer network, 9542 cycles were required to discover at least one path between every pair of source and destination. Another overhead is the router resources required. Single router resource utilization targeting Virtex-7 FPGA (xc7vx485tffg1761-2) is shown in Table I. The base router, which was designed for regular mesh topology comprised of 5 ports. Resource utilization for a minimum and maximum port count router for the multi-layer topology under test is shown in Table 7.5. Two extra ports are added, one to connect to the local processing tile and another referred to as a drop port. A drop port is added to drop failed ant packets. Further architecture optimization is possible here to avoid the drop port since buffers are resource heavy. In this section, we evaluated DREAM multi-layered NoC with its supporting framework and the results were published in [106].

Table 7.5: Resource utilization comparison of a single router [106]

ACO based	Total (10 Port)	Total (5 Port)
LUT	7868	6423
FF	5801	5331

7.8 Summary

In this chapter, the evaluations of the DREAM NoC along with its supporting framework and its multi-layered extension is presented. Section 7.1.1 described an overview of the test bench developed. In section 7.2, the ability of the DREAM routing to discover minimal and non-minimal paths in regular and irregular networks was discussed. Such features are beneficial to ensure load balancing and potential of using DREAM routing in traffic isolation within partitions was investigated. The learning time overhead was analysed in section 7.2.1 along with preliminary resource utilization. To reduce this learning time overhead, dynamically changing the alpha value which represents exploitation vs exploration of routes was evaluated in section 7.3. The DREAM NoC and supporting framework was evaluated in section 7.4 and latency and throughput results were presented. The results showed improved performance when compared to other topology agnostic routing schemes. We also tested it under MPEG4 application test case as discussed in section 7.4.2. We also evaluated adaptive Block-Based multicast routing technique in section 7.5 and compared the address overhead scalability to traditional destination encoding schemes. At the topology level, multi-layer networks was proposed targeting mixed critical NoCs and supporting adaptive congestion aware routings was evaluated in section 7.6. The proposed adaptive routing showed improved performance against the static routing tecniques. The DREAM NoC was further developed to support such multi-layer NoCs as well. The reinforcement learning enabled DREAM routing was evaluated and performed better when compared against other static and dynamic routing techniques as seen in section 7.7. The chapter also presented the overhead in terms of resource utilization and learning time overhead.

8 Conclusion and Future Work

8.1 Conclusion

Recently, there have been multiple changes in the field of semiconductors that have impacted the microprocessor industry. The ending of Dennard scaling around 2004 led to the microprocessor industry to replace a single inefficient processor with multiple efficient processors or cores. Additionally, the prediction in Moore's Law where the number of transistors per chip would double every two years, recently ended. But there is always an ever increasing demand for computational services. This has resulted in the development of complex systems to improve performance. The increase in integration density has resulted in MPSoC platforms evolving to contain 100s of processing cores. Increasing the number of cores results in its own set of challenges. Focusing on communication, researchers identified that traditional bus-based communication was becoming a limiting factor for performance. Which led to the development of NoCs, which is a scalable solution for the increasing communication requirements of complex MPSoCs. NoCs are a modular, scalable and flexible communication infrastructure.

Realtime applications are a major portion of applications in the embedded systems field. Realtime computation and communication need to meet certain stringent deadlines, which impose special constraints on such MPSoC platforms. An increasing trend when designing realtime embedded systems is the development of MCS where components of different criticality are integrated on the same computing platform to meet stringent non-functional requirements. With the increase in the number of cores and applications sharing resources, the on-chip communicaton is becoming a critically shared resource. NoCs in mixed criticality

systems have their own range of challenges. They include providing support for QoS, where guarantees in terms of latency and throughput need to be ensured for critical applications. Realtime MPSoCs are subjected to varying workloads and application requirements at runtime, motivating the need to support runtime adaptive NoCs. There is potential in making a mixed criticality NoC more adaptable at runtime to improve overall performance. An ongoing research area is developing runtime adaptive NoCs while ensuring realtime guarantees are met.

In this work, the aim is to address the above challenges and present an adaptive NoC targeting mixed criticality applications. The thesis proposes, develops and evaluates adaptive features at the routing and topology level to improve performance like latency and throughput. The NoC also provides QoS service, guaranteeing throughput and latency for critical applications. An overview of essential contributions in this work is summarized below.

DREAM NoC: A **D**istributed **R**einforcement learning **E**nabled **A**daptive **M**ixed-Critical Network-on-Chip (DREAM NoC) was proposed in this work. DREAM is a distributed and self-learning NoC which uses a topology agnostic, reinforcement learning enabled routing algorithm based on the Ant Colony Optimization (ACO) metaheuristic [115]. The routing scheme discovers better paths over time based on traffic fluctuations and by monitoring the network. The thesis presents a routing scheme targeting mixed criticality NoCs based on the AntNet model [101, 41] with focus on the reinforcement learning aspect. A contribution in this work includes simplification of complex computations and methodologies in the base model used, to be feasible for hardware implementation while still maintaining the inherent flexibility the routing scheme offers. *In this work, identifying the potential of using such a topology agnostic, reinforcement learning enabled routing scheme to develop adaptive NoCs to aid in flexible spatial isolation for mixed criticality systems is presented* A hardware friendly router implementing the routing scheme as a synthesizable HDL model is shown in this work.

DREAM Framework for spatial isolation in MCS: *In this work, the DREAM framework comprising of learning phase and application phase to utilise the DREAM*

routing mechanism effectively to aid in spatial isolation of NoC resources is presented. Learning phase was assigned to build preliminary routing tables at each router in a *distributed* manner and investigations were conducted to decrease this learning time overhead as well. During the application runtime, the routing tables built are updated periodically over time when new better routes are discovered by the routing algorithm. Test benches in HDL was developed to evaluate the NoC with this framework, for a range of regular and irregular partitions under different traffic scenarios. Evaluations using an MPEG4 application use case was also presented.

Multi-Layered NoCs for MCS: In this work, a multi-layered network running adaptive congestion-aware routing to decouple mixed-critical traffic is proposed. The aim here was to reduce inter-application influence. The topology and supporting routing schemes were are also developed in HDL. The DREAM NoC was extended to support such multi-layered topologies as well and performance was compared against static and dynamic routing schemes.

Adaptive multicast techniques: A flexible and scalable multicast routing technique called as Block-Based routing is proposed. The aim here was to improve traditional multicast routing by making it more scalable, adaptive and flexible. Here, a block of destination nodes can be defined at runtime.The proposed solution scales better as the bit vector addressing encoding used is restricted to the block of nodes. It is more flexible as bit vector length can be changed dynamically at runtime.

The router design used in all the above contributions is a base router, comprising of state of the art features. It is a configurable pipeline based design utilizing wormhole packet switching with virtual channels and credit-based flow control. Deadlock and livelock avoidance mechanisms were developed for the proposed adaptive routing techniques. The router supports both BE and GS communication which is vital for critical applications. A common overhead of the proposed techniques is increased resource utilization. This is primarily due to the custom computation units, table based routing schemes and increased port count in the DREAM NoC. This port count is further extended in the multi-layer NoC version.

Hardware optimizations are possible and are part of future work as outlined in the next section.

8.2 Future Work

The proposed DREAM NoC, supporting framework, adaptive routing schemes and the multi-layer extension provide a comprehensive basis for future adaptive mixed criticality NoCs. There is potential for future work. An overview of general improvements possible, especially addressing the overhead of resource utilization is provided next.

Table based routing has an inherent resource overhead when the network is scaled up. When the number of cores increases in the network, the tables do not scale well in terms of area overhead. For larger network sizes, if the maximum possible number of nodes in a partition is known, then the size of the routing/pheromone tables can be fixed accordingly so as to avoid having all the source destination pair information stored in the table. An alternative is to apply a window based solution described in [112]. As part of future work, further hardware optimizations can be investigated to reduce resource consumption by reducing input buffer overhead. In the current design, each port has exclusive routing module and sharing the routing unit implementing the DREAM routing across multiple ports within the router can decrease overall resource overhead. For routers with high port counts, it would be beneficial to investigate the possibility of clock gating for unused resources to decrease power consumption.

Next, a more general extension as part of future work is presented. Here the focus is more on the hardware architecture of the NoC. It would be beneficial to develop a supporting software-based management of the proposed concepts and features. Developing an operating system for example, to manage the reinforcement learning based DREAM routing. Some areas of future work include, incorporating OS features of transmitting relevant information to the processing tiles to trigger learning and application phases of the DREAM framework, communicating to

the processing tiles of the border routers etc. The alpha parameter representing the exploration vs exploitation of routes can be investigated further in tandem with a supporting operating system.

Though the work here focused in aiding in flexible spatial isolation, the other advantages of the reinforcement learning based DREAM NoC is increasing reliability in the network when faced with network component failure. Due to the runtime discovery of better paths, alternate paths can be discovered in case of unpredicted change in the network state. This feature can be investigated further to develop efficient fault tolerant mechanisms. In this work, primary 2D mesh based and multi-layer topologies were investigated. Evaluating on 3D network topologies opens a range of possibility and opportunities. In this work, the focus was on wired NoCs. Investigating the proposed concepts on wireless NoCs offers an interesting research area. In wireless NoCs, on-chip antennas are integrated at the routers to transmit the packets wirelessly across the network. Such wireless NoCs have the advantage of high bandwidth but care must be taken regarding data loss and error protection.

List of Figures

List of Tables

List of Publications

Own Publications

1. N. Anantharajaiah, F. Lesniak, T. Harbaum and J. Becker, "Reinforcement Learning Enabled Multi-Layered NoC for Mixed Criticality Systems," 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 2023, pp. 38-44

2. N. Anantharajaiah, Y. Xu, F. Lesniak, T. Harbaum and J. Becker, "DREAM: Distributed Reinforcement Learning Enabled Adaptive Mixed-Critical NoC," 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Foz do Iguacu, Brazil, 2023, pp. 1-6

3. N. Anantharajaiah and J. Becker, "Adaptive Exploration Based Routing for Spatial Isolation in Mixed Criticality Systems," 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 2022, pp. 174-180,

4. N. Anantharajaiah, F. Knopf and J. Becker, "Ant Colony Optimization Based NoCs for Flexible Spatial Isolation in Mixed Criticality Systems," 2021 IEEE 34th International System-on-Chip Conference (SOCC), 2021, pp. 248-253,

5. N. Anantharajaiah, Zhang Z., Becker J. (2021) “Multi-layered NoCs with Adaptive Routing for Mixed Criticality Systems”. In: Derrien S., Hannig

F., Diniz P.C., Chillet D. (eds) Applied Reconfigurable Computing. Architectures, Tools, and Applications. ARC 2021. Lecture Notes in Computer Science, vol 12700. Springer, Cham.

6. N. Anantharajaiah; Kempf, F.; Masing, L.; Lesniak, F. M.; Becker, J. Dynamic and scalable runtime block-based multicast routing for networks on chips. 2019. Proceedings of the 12th International Workshop on Network on Chip Architectures (NoCArc 2019), Columbus, OH

Co-Author

1. A. Chu et al., "LETSCOPE: Lifecycle Extensions Through Software-Defined Predictive Control of Power Electronics," IEEE EUROCON 2023 - 20th International Conference on Smart Technologies, Torino, Italy, 2023, pp. 665-670

2. Fabian Lesniak, Nidhi Anantharajaiah, Tanja Harbaum, and Juergen Becker. 2023. Non-Intrusive Runtime Monitoring for Manycore Prototypes. In Proceedings of the DroneSE and RAPIDO: System Engineering for constrained embedded systems (RAPIDO '23). Association for Computing Machinery, New York, NY, USA, 31–38.

3. Jesse Barreto de Barros, Nidhi Anantharajaiah, Mauricio Ayala-Rincón, Carlos Humberto Llanos, Jürgen Becker, "The impact of formulation of cost function in Task Mapping Problem on NoCs using bio-inspired based-metaheuristics", Microprocessors and Microsystems, Volume 94, 2022, 104668, ISSN 0141-9331

4. J. B. de Barros, N. Anantharajaiah, M. Ayala-Rincón, C. H. Llanos and J. Becker, "A Study of the Impact of Formulation of Cost Function in Task Mapping Problem on NoCs," 2020 IEEE Nordic Circuits and Systems Conference (NorCAS), 2020, pp. 1-7

5. F. Kempf, N. Anantharajaiah, L. Masing and J. Becker, "A Network on Chip Adapter for Real-Time and Safety-Critical Applications," 2019 32nd IEEE International System-on-Chip Conference (SOCC), 2019, pp. 39-44

6. L. Masing, A. Srivatsa, F. Kreß, N. Anantharajaiah, A. Herkersdorf and J. Becker, "In-NoC Circuits for Low-Latency Cache Coherence in Distributed Shared-Memory Architectures," 2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2018, pp. 138-145

Supervised Student Research

1. Yingnan Jiang, *Investigating ACO based routing for self-adaptive Network-on-Chips*, Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2022

2. Yunhe Xu, *Investigating Adaptive Routing for Isolating Traffic in Mixed Criticality NoCs*, Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2021

3. Dirk Lenhardt, *Performance Comparison between the SXP and PCI Express Protocol with a Refinement Approach of SXP with Security*, Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2021

4. Zhe Zhang, *Improving Quality of Service in Adaptive Networks-on-Chip for Mixed-Criticality System*, Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2020

5. Felix Knopf, *Design and Implementation of an ACO Based Router for Mixed Criticality NoCs*, Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2020

6. Ali Acheche, *Design and Implementation of a Self-Adaptive NoC*, Bachelor Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2019

7. Felix Knopf, *Introduction and Analysis of Methods for Adaptive Networks-on-Chip*, Seminar Eingebettete System, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2018

8. Tim Etkin, *Quality of Service in Adaptive Networks-on-Chip*, Seminar Eingebettete System, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2019

Bibliography

[1] Conor Cunningham. "Survival of the Fittest", Encyclopedia Britannica, 8 Jun. 2023.

[2] David A. Patterson John L. Hennessy. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, 6th edition edition, November 23, 2017.

[3] Versal:The First Adaptive Compute Acceleration Platform (ACAP). `https://docs.xilinx.com/v/u/en-US/wp505-versal-acap`, White Paper, WP505 (v1.1.1), September 29, 2020.

[4] Ampereone 64-bit multi-core processors. `https://amperecomputing.com/briefs/ampereone-family-product-brief`, (accessed:Nov.02 2023).

[5] International Technology Roadmap for Semiconductors 2.0 (ITRS), Executive Report. `https://www.semiconductors.org/wp-content/uploads/2018/06/0_2015-ITRS-2.0-Executive-Report-1.pdf`, ITRS 2015.

[6] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 14–27, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Jieming Yin, Zhifeng Lin, Onur Kayiran, Matthew Poremba, Muhammad Shoaib Bin Altaf, Natalie Enright Jerger, and Gabriel H. Loh. Modular routing design for chiplet-based systems. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 726–738, 2018.

[8] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, 2002.

[9] Salma Hesham, Jens Rettkowski, Diana Goehringer, and Mohamed A. Abd El Ghany. Survey on real-time networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1500–1517, 2017.

[10] Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6), nov 2017.

[11] Mariem Turki and Davide Bertozzi. An interconnect-centric approach to the flexible partitioning and isolation of many-core accelerators for fog computing. In *2019 XXXIV Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, 2019.

[12] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), may 2008.

[13] Nidhi Anantharajaiah, Yunhe Xu, Fabian Lesniak, Tanja Harbaum, and Juergen Becker. Dream: Distributed reinforcement learning enabled adaptive mixed-critical noc. In *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 1–6, 2023.

[14] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[15] Luca Benini Giovanni De Micheli. *Networks on Chips Technology and Tools*. Morgan Kaufmann, 1st edition, 2006.

[16] D. Flynn. Amba: enabling reusable on-chip designs. *IEEE Micro*, 17(4):20–27, 1997.

[17] L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *Computer*, 35(1):70–78, 2002.

[18] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, 2001.

[19] Hannu Tenhunen Axel Jantsch. *Networks on Chip*. Springer New York, NY, 1st edition, 2003.

[20] Jan Heisswolf. *A Scalable and Adaptive Network on Chip for Many-Core Architectures*. PhD thesis, 2014.

[21] Nidhi Anantharajaiah, Felix Knopf, and Juergen Becker. Ant colony optimization based nocs for flexible spatial isolation in mixed criticality systems. In *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, pages 248–253, 2021.

[22] Thomas Moscibroda and Onur Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 196–207, New York, NY, USA, 2009. Association for Computing Machinery.

[23] Ammar Karkar, Terrence Mak, Kin-Fai Tong, and Alex Yakovlev. A survey of emerging interconnects for on-chip efficient multicast and broadcast in many-cores. *IEEE Circuits and Systems Magazine*, 16(1):58–72, 2016.

[24] Stephanie Friederich, Niclas Lehmann, and Jürgen Becker. Adaptive bandwidth router for 3d network-on-chips. In Vanderlei Bonato, Christos Bouganis, and Marek Gorgon, editors, *Applied Reconfigurable Computing*, pages 352–360, Cham, 2016. Springer International Publishing.

[25] D. Wiklund and Dake Liu. Socbus: switched network on chip for hard real time embedded systems. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 8 pp.–, 2003.

[26] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *IEEE Design & Test of Computers*, 22(5):414–421, 2005.

[27] E. Rijpkema, K.G.W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 350–355, 2003.

[28] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 890–895 Vol.2, 2004.

[29] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration*, 38(1):69–93, 2004.

[30] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[31] Jan Heißwolf. *A Scalable and Adaptive Network on Chip for Many-Core Architectures*. PhD thesis, Karlsruhe Institute of Technology, 2014.

[32] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[33] Mayank Parasar, Natalie Enright Jerger, Paul V. Gratz, Joshua San Miguel, and Tushar Krishna. Swap: Synchronized weaving of adjacent packets for network deadlock resolution. MICRO '52, page 873–885, New York, NY, USA, 2019. Association for Computing Machinery.

[34] Jan Heisswolf, Maximilian Singh, Martin Kupper, Ralf König, and Jürgen Becker. Rerouting: Scalable noc self-optimization by distributed hardware-based connection reallocation. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2013.

[35] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018.

[36] G. A. Di Caro. *Ant Colony Optimization and its application to adaptive routing in telecommunication networks*. PhD thesis, 2004.

[37] Felix Knopf. Design and implementation of an aco based router for mixed criticality nocs. 2020. Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV).

[38] S Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76:579, 01 1989.

[39] Bert Hölldobler and Edward O. Wilson. *The Ants*. Belknap Press of Harvard University Press, Cambridge, Mass., 1990.

[40] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[41] G.A. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research (JAIR)*, 9:317–365, 1998.

[42] Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. *Invasive Computing: An Overview*, pages 241–268. Springer New York, New York, NY, 2011.

[43] Nidhi Anantharajaiah, Tamim Asfour, Michael Bader, Lars Bauer, Jürgen Becker, Simon Bischof, Marcel Brand, Hans-Joachim Bungartz, Christian Eichler, Khalil Esper, Joachim Falk, Nael Fasfous, Felix Freiling, Andreas Fried, Michael Gerndt, Michael Glaß, Jeferson Gonzalez, Frank Hannig,

Christian Heidorn, Jörg Henkel, Andreas Herkersdorf, Benedict Herzog, Jophin John, Timo Hönig, Felix Hundhausen, Heba Khdr, Tobias Langer, Oliver Lenke, Fabian Lesniak, Alexander Lindermayr, Alexandra Listl, Sebastian Maier, Nicole Megow, Marcel Mettler, Daniel Müller-Gritschneder, Hassan Nassar, Fabian Paus, Alexander Pöppl, Behnaz Pourmohseni, Jonas Rabenstein, Phillip Raffeck, Martin Rapp, Santiago Narvez Rivas, Mark Sagi, Franziska Schirrmacher, Ulf Schlichtmann, Florian Schmaus, Wolfgang Schröder-Preikschat, Tobias Schwarzer, Mohammed Bakr Sikal, Bertrand Simon, Gregor Snelting, Jan Spieck, Akshay Srivatsa, Walter Stechele, Jürgen Teich, Furkan Turan, Isaas A. Comprs Urea, Ingrid Verbauwhede, Dominik Walter, Thomas Wild, Stefan Wildermann, Mario Wille, Michael Witterauf, and Li Zhang. *Invasive Computing*. 2022.

[44] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.

[45] M.B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim. Evaluation of the raw microprocessor: an exposed-wire-delay architecture for ilp and streams. In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, pages 2–13, 2004.

[46] Carl Ramey. Tile-gx100 manycore processor: Acceleration interfaces and architecture. In *2011 IEEE Hot Chips 23 Symposium (HCS)*, pages 1–21, 2011.

[47] Sagheer Ahmad, Sridhar Subramanian, Vamsi Boppana, Shankar Lakka, Fu-Hing Ho, Tomai Knopp, Juanjo Noguera, Gaurav Singh, and Ralph Wittig. Xilinx first 7nm device: Versal ai core (vc1902). In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–28, 2019.

[48] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Bala Jayadev, Jeff Cuppett, Abbas Morshed, Brian Gaide, and Ygal Arbel. Versal network-on-chip

(noc). In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 13–17, 2019.

[49] Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller v1.0, LogiCORE IP Product Guide Vivado Design Suite. `https://docs.xilinx.com/r/en-US/pg313-network-on-chip`, PG313 (v1.0) November 1, 2023.

[50] F. A. Samman, T. Hollstein, and M. Glesner. Runtime contention and bandwidth-aware adaptive routing selection strategies for networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1411–1421, 2013.

[51] M. A. Al Faruque, T. Ebi, and J. Henkel. Adnoc: Runtime adaptive network-on-chip architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(2):257–269, 2012.

[52] Konstantinos Tatas and Chrysostomos Chrysostomou. Hardware implementation of dynamic fuzzy logic based routing in network-on-chip. *Microprocessors and Microsystems*, 52:80 – 88, 2017.

[53] E. Chang, H. Hsin, S. Lin, and A. Wu. Path-congestion-aware adaptive routing with a contention prediction scheme for network-on-chip systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(1):113–126, 2014.

[54] Aqib Javed, Jim Harkin, Liam McDaid, and Junxiu Liu. Minimising impact of local congestion in networks-on-chip performance by predicting buffer utilisation. In *31st Irish Signals and Systems Conference*, pages 1–6, United States, 5 2020. IEEE.

[55] G. Ascia, V. Catania, M. Palesi, and D. Patti. Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. *IEEE Transactions on Computers*, 57(6):809–820, 2008.

[56] P. Gratz, B. Grot, and S. W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 203–214, 2008.

[57] R. Manevich, I. Cidon, a. kolodny, and I. Walter. Centralized adaptive routing for NoCs. *IEEE Computer Architecture Letters*, 9(2):57–60, 2010.

[58] Sebastian Tobuschat. *Predictable and Runtime-Adaptable Network-On-Chip for Mixed-critical Real-time Systems*. PhD thesis, Technische Universität Braunschweig, 2019.

[59] S. Liu, T. Chen, L. Li, X. Li, M. Zhang, C. Wang, H. Meng, X. Zhou, and Y. Chen. Freerider: Non-local adaptive network-on-chip routing with packet-carried propagation of congestion information. *IEEE Transactions on Parallel and Distributed Systems*, 26(8):2272–2285, 2015.

[60] T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel. Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, pages 527–534, 2007.

[61] M. Nickray, M. Dehyadgari, and A. Afzali-kusha. Adaptive routing using context-aware agents for networks on chips. In *2009 4th International Design and Test Workshop (IDT)*, pages 1–6, 2009.

[62] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, J. Plosila, and P. Liljeberg. Adaptive reinforcement learning method for networks-on-chip. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 236–243, 2012.

[63] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, J. Plosila, and P. Liljeberg. Optimized q-learning model for distributing traffic in on-chip networks. In *2012 IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA)*, pages 1–8, 2012.

[64] M. Ebrahimi, M. Daneshtalab, F. Farahnakian, J. Plosila, P. Liljeberg, M. Palesi, and H. Tenhunen. Haraq: Congestion-aware learning model for

highly adaptive routing algorithm in on-chip networks. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 19–26, 2012.

[65] Muhammad Athar Javed Sethi, Fawnizu Azmadi Hussin, and Nor Hisham Hamid. Bio-inspired network on chip having both guaranteed throughput and best effort services using fault-tolerant algorithm. *IEEJ Transactions on Electrical and Electronic Engineering*, 13(8):1153–1162, 2018.

[66] Marcelo Daniel Berejuck. An overview about networks-on-chip with multicast support. *CoRR*, abs/1610.00751, 2016.

[67] M. Ebrahimi, M. Daneshtalab, M. H. Neishaburi, S. Mohammadi, A. Afzali-Kusha, J. Plosila, and H. Tenhunen. An efficent dynamic multicast routing protocol for distributing traffic in nocs. In *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009.

[68] P. Abad, V. Puente, L. G. Menezo, and J. Angel Gregorio. Adaptive-tree multicast: Efficient multidestination support for cmp communication substrate. *IEEE Transactions on Parallel and Distributed Systems*, 23(11), Nov 2012.

[69] M. P. Malumbres, Jose Duato, and Joseph Torrellas. An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP '96)*, SPDP '96, Washington, DC, USA, 1996. IEEE Computer Society.

[70] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, Washington, DC, USA, 2008. IEEE Computer Society.

[71] S. Rodrigo, J. Flich, J. Duato, and M. Hummel. Efficient unicast and multicast support for cmps. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*, Nov 2008.

[72] Xu Liu, Jiang Jiang, Yongxin Zhu, Chang Wang, and Xing Han. A load-aware broadcast scheme supporting rectangular regions for many-core processors. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, volume 01, Dec 2015.

[73] Lei Wang, Yuho Jin, Hyungjun Kim, and Eun Jung Kim. Recursive partitioning multicast: A bandwidth-efficient routing for networks-on-chip. In *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, Washington, DC, USA, 2009. IEEE Computer Society.

[74] Y. H. Kang, J. Sondeen, and J. Draper. Implementing tree-based multicast routing for write invalidation messages in networks-on-chip. In *2009 52nd IEEE International Midwest Symposium on Circuits and Systems*, Aug 2009.

[75] Jan Heißwolf, Andreas Weichslgartner, Aurang Zaib, Ralf König, Thomas Wild, Andreas Herkersdorf, Jürgen Teich, and Jürgen Becker. Hardware supported adaptive data collection for networks on chip. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 153–162, 2013.

[76] Andreas Lankes, Thomas Wild, and Andreas Herkersdorf. Hierarchical nocs for optimized access to shared memory and io resources. In *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pages 255–262, 2009.

[77] Simon J. Hollis, Chris Jackson, Paul Bogdan, and Radu Marculescu. Exploiting emergence in on-chip interconnects. *IEEE Transactions on Computers*, 63(3):570–582, 2014.

[78] Sebastian Tobuschat and Rolf Ernst. Providing throughput guarantees in mixed-criticality networks-on-chip. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 292–297, 2017.

[79] Adele Maleki, Hamidreza Ahmadian, and Roman Obermaisser. Fault-tolerant and energy-efficient communication in mixed-criticality networks-on-chips. In *2018 IEEE Nordic Circuits and Systems Conference (NORCAS):*

NORCHIP and International Symposium of System-on-Chip (SoC), pages 1–7, 2018.

[80] Serhiy Avramenko, Siavoosh Payandeh Azad, Stefano Esposito, Behrad Niazmand, Massimo Violante, Jaan Raik, and Maksim Jenihhin. Qosinnoc: Analysis of qos-aware noc architectures for mixed-criticality applications. In *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 67–72, 2018.

[81] Mourad Dridi, Stéphane Rubini, Mounir Lallali, Martha Johanna Sepúlveda Flórez, Frank Singhoff, and Jean-Philippe Diguet. Design and multi-abstraction-level evaluation of a noc router for mixed-criticality real-time systems. *J. Emerg. Technol. Comput. Syst.*, 15(1), feb 2019.

[82] Jose Flich, Tor Skeie, Andres Mejia, Olav Lysne, Pedro Lopez, Antonio Robles, Jose Duato, Michihiro Koibuchi, Tomas Rokicki, and Jose Carlos Sancho. A survey and evaluation of topology-agnostic deterministic routing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):405–425, 2012.

[83] Yunhe Xu. Investigating adaptive routing for isolating traffic in mixed criticality nocs. 2021. Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV).

[84] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C.P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, 1991.

[85] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and Wen-King Su. Myrinet: a gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[86] David Garcia and William Watson. Servernet ™ ii. In Sudhakar Yalamanchili and José Duato, editors, *Parallel Computer Routing and Communication*, pages 119–135, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[87] José Carlos Sancho, Antonio Robles, and José Duato. A new methodology to compute deadlock-free routing tables for irregular networks. In Babak Falsafi and Mario Lauria, editors, *Network-Based Parallel Computing. Communication, Architecture, and Applications*, pages 45–60, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[88] José Flich, Samuel Rodrigo, and José Duato. An efficient implementation of distributed routing algorithms for nocs. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pages 87–96, 2008.

[89] Samuel Rodrigo, José Flich, Antoni Roca, Simone Medardoni, Davide Bertozzi, Jesús Camacho, Federico Silla, and José Duato. Cost-efficient on-chip routing implementations for cmp and mpsoc systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):534–547, 2011.

[90] M. Daneshtalab, A. Sobhani, A. Afzali-Kusha, O. Fatemi, and Z. Navabi. NoC hot spot minimization using AntNet dynamic routing algorithm. In *IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06)*, pages 33–38, 2006.

[91] En-Jui Chang, Chih-Hao Chao, Kai-Yuan Jheng, Hsien-Kai Hsin, and An-Yeu Wu. ACO-based cascaded adaptive routing for traffic balancing in NoC systems. In *The 2010 International Conference on Green Circuits and Systems*, pages 317–322, 2010.

[92] En-Jui Chang and An-Yeu Wu. Overview of high-efficiency ant colony optimization (ACO)-based adaptive routings for traffic balancing in network-on-chip systems. In *2017 IEEE 12th International Conference on ASIC (ASICON)*, pages 80–83, 2017.

[93] K. Su, H. Hsin, E. Chang, and A. Wu. Aco-based deadlock-aware fully-adaptive routing in network-on-chip systems. In *2012 IEEE Workshop on Signal Processing Systems*, pages 209–214, 2012.

[94] H. Hsin, E. Chang, C. Lin, and A. Wu. Ant colony optimization-based fault-aware routing in mesh-based network-on-chip systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(11):1693–1705, 2014.

[95] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, page 671–678, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[96] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and J. Plosila. Bi-lcq: A low-weight clustering-based q-learning approach for nocs. *Microprocessors and Microsystems*, 38(1):64–75, 2014.

[97] Kamil Khan and Sudeep Pasricha. A reinforcement learning framework with region-awareness and shared path experience for efficient routing in networks-on-chip. *IEEE Design & Test*, 40(6):76–85, 2023.

[98] Samuel P. M. Choi and Dit-Yan Yeung. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 945–951, Cambridge, MA, USA, 1995. MIT Press.

[99] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *J. Artif. Int. Res.*, 9(1):317–365, dec 1998.

[100] Nidhi Anantharajaiah, Fabian Kempf, Leonard Masing, Fabian Marc Lesniak, and Juergen Becker. Dynamic and scalable runtime block-based multicast routing for networks on chips. In *Proceedings of the 12th International Workshop on Network on Chip Architectures*, NoCArc '19, New York, NY, USA, 2019. Association for Computing Machinery.

[101] G. A. Di Caro. *Ant Colony Optimization and its application to adaptive routing in telecommunication networks*. PhD thesis, Polytechnic School, Université Libre de Bruxelles, 2004.

[102] Dmitri Vainbrand and Ran Ginosar. Network-on-chip architectures for neural networks. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '10, Washington, DC, USA, 2010. IEEE Computer Society.

[103] Akshay Srivatsa, Sven Rheindt, Thomas Wild, and Andreas Herkersdorf. Region based cache coherence for tiled mpsocs. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 286–291, 2017.

[104] Zhe Zhang. Improving quality of service in adaptive networks-on-chip for mixed-criticality system. 2020. Master Thesis, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV).

[105] Nidhi Anantharajaiah, Zhe Zhang, and Juergen Becker. Multi-layered nocs with adaptive routing for mixed criticality systems. In Steven Derrien, Frank Hannig, Pedro C. Diniz, and Daniel Chillet, editors, *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, pages 125–139. Springer International Publishing, 2021.

[106] T.Harbaum N.Anantharajaiah, F.Lesniak and J.Becker. Reinforcement learning enabled multi-layered noc for mixed criticality systems. In *16th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-2023), Singapore (Accepted)*, 2023.

[107] Sebastian Tobuschat and Rolf Ernst. Efficient latency guarantees for mixed-criticality networks-on-chip. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 113–122, 2017.

[108] H. Xu and A. Burns. A semi-partitioned model for mixed criticality systems. *Journal of Systems and Software*, 150:51–63, 2019.

[109] Nidhi Anantharajaiah and Juergen Becker. Adaptive exploration based routing for spatial isolation in mixed criticality systems. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 174–180, 2022.

[110] Frank Simon. *Algebraic Methods for Computing the Reliability of Networks*. PhD thesis, 2012. Dissertation, Doctor Rerum Naturalium (Dr.rer.nat.), Fakultät Mathematik und Naturwissenschaften der Technischen Universität Dresden, Avaliable at https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-101154.

[111] Jan Heisswolf, Simon Bischof, Michael Rückauer, and Jürgen Becker. Efficient memory access in 2d mesh noc architectures using high bandwidth routers. In *2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2013.

[112] En-Jui Chang, Chih-Hao Chao, Kai-Yuan Jheng, Hsien-Kai Hsin, and An-Yeu Wu. Aco-based cascaded adaptive routing for traffic balancing in noc systems. In *The 2010 International Conference on Green Circuits and Systems*, pages 317–322, 2010.

[113] Donald G. Bailey. Space efficient division on fpgas. In *Electronics New Zealand Conference 2006*, 2006.

[114] Xiaohang Wang, Mei Yang, Yingtao Jiang, and Peng Liu. Power-aware mapping for network-on-chip architectures under bandwidth and latency constraints. In *2009 Fourth International Conference on Embedded and Multimedia Computing*, pages 1–6, 2009.

[115] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477 Vol. 2, 1999.