Ministry of Higher Education and Scientific Research
University of Saad Dahleb - BLIDA 1
Faculty of Sciences
Department of Informatics

Performance Evaluation of Networks On-chip Topologies

Report submitted for the fulfillment of the Master degree
Domain: MI
Affiliation: Informatics
Specialisation: Computer systems and networks

Adel-Salah Ould-Khaoua and Hichem Terranti

Supervisor: Prof. Mohamed Ould-Khaoua

Academic year: 2019/2020

Acknowledgment

We would like to greatly thank our supervisor: Prof. Mohamed Ould-Khaoua, for his invaluable advice and guidance during the course of this project.

Our sincere thanks also go to the members of the jury for their interest in our project by agreeing to examine our work and enrich it with their suggestions and recommendations.

Our gratitude is due to our instructors who taught us during our License and Master degrees.

Finally, we would like to thank everyone who participated either directly or indirectly in the achievement of this work.

Abstract

The main objective of this project is to evaluate the performance of some well-known topologies that have been proposed for Networks-on-Chip including the mesh and torus. Whereas existing studies have focused on the graph-theoretical merits of such topologies, our study examines the performance of networks-on-chip taking into account the constraints imposed by implementation technology. The most relevant constraint for networks-on-chip is the wiring density of the chip. To achieve our goal, we have developed a simulation model using the discrete-event simulation technique. Extensive simulation experiments have been performed and the collected results indicate that while the bidirectional torus has superior performance when technological constraints are ignored due to its richer connectivity, its performance degrades considerably compared to the mesh once technological constraints are considered. Our results also indicate that the 2D topologies are more suitable for networks-on-chip than their 3D counterparts as they are a better fit for practical implementations.

Résumé

L'objectif principal de ce projet est d'évaluer les performances de certaines topologies bien connues pour les réseaux sur puce, y compris le mesh et le torus alors que les topologies existants se sont concentrés sur les propriétés théoriques des graphes de telles topologies, notre étude examine les performance des réseaux sur puce en tenant compte des contraintes imposées par la technologie de mise en œuvre. La contrainte la plus importante pour les réseaux sur puce est la densité de câblage de la puce. Pour atteindre notre objectif, nous avons développé un modèle de simulation utilisant la technique de simulation par événements discrets. De nombreuses expériences de simulation ont été réalisées et les résultats collectés ont indiqué que si le torus bidirectionnel a des performances supérieures lorsque les contraintes technologiques sont ignorées, ses performances se dégradent considérablement par rapport au mesh une fois les contraintes technologiques incluses. Les résultats indiquent également que les topologies 2D sont bien mieux adaptées aux réseaux sur puce que leurs homologues 3D car elles conviennent mieux aux implémentations pratiques.

ملخص

الهدف الرئيسي من هذا المشروع هو تقييم أداء بعض الطوبولوجيا المعروفة للشبكات على الشريحة بما في ذلك mesh و torus في حين ركزت الهياكل الحالية على مزايا الخصائص النظرية للرسم البياني لهذه الطوبولوجيا، تفحص دراستنا أداء الشبكات على الشريحة مع مراعاة القيود التي تفرضها تكنولوجيا التنفيذ. أكثر القيود ذات الصلة بالنسبة للرقاقة هي كثافة الأسلاك الخاصة بها. لتحقيق هدفنا ، قمنا بتطوير نموذج محاكاة باستخدام تقنية محاكاة الحدث المنفصل. تم إجراء تجارب محاكاة واسعة النطاق ، وقد أشارت النتائج التي تم جمعها إلى أنه في حين أن bidirectional torus يتمتع بأداء متفوق عند تجاهل القيود التكنولوجية ، فإن أداءه يتدهور إلى حد كبير مقارنة مع mesh بمجرد إدراج القيود التكنولوجية. تشير النتائج أيضًا إلى أن الهياكل ثنائية الأبعاد 2D مناسبة بشكل أفضل للشبكات على الشريحة من نظيراتها ثلاثية الأبعاد 3D لأنها مناسبة بشكل أفضل للتطبيقات العملية.

Contents

List of abbreviations

BFT	Butterfly-Fat-Tree
FT	Fat-Tree
NoC	Network-on-Chip
PE	Processing Element
R	Router
SoC	Systems-On-Chip
SONoC	Square Octagon Network-on-Chip

List of symbols

B_{mesh}	Channel bandwidth of the mesh topology
$B_{unitorus}$	Channel bandwidth of the unidirectional torus topology
$B_{bitorus}$	Channel bandwidth of the bidirectional torus topology
D	Number of hops between the sender and destination nodes
k	Number of nodes per dimension
L	Message length
N	Total number of nodes in the network
n	Number of dimensions of a topology
p	Probability of sending a message to the hotspot node
R	Response time
$\bar{R}$	Response time of one batch
s	Standard deviation
$t_{arrival}$	Time of arrival of the message in the sender node
t_{now}	Current time in the simulation
Th	Throughput
W_{2D}	Bisection width of the 2d topologies
W_{3D}	Bisection width of the 3d topologies
W_{mesh}	Bisection width of the mesh topology
$W_{unitorus}$	Bisection width of the unidirectional torus topology
$W_{bitorus}$	Bisection width of the bidirectional torus topology
$\bar{x}$	Average of a sample
λ	Arrival rate

List of figures

List of tables

Introduction

Much research and development have been aimed at increasing processing power by incorporating concurrent computations such as parallel processing. This has largely been aided by Moore's law which has been the driving force behind improvements in integrated chip technology for the past five decades. Although the exponential trend is expected to slow down considerably within the next few years, it is still currently in force [1]. VLSI technology has matured to such a point that it has enabled a paradigm shift that allowed the introduction of entire systems consisting of a large number of processing elements of different computation capabilities (CPUs, GPUs, DSPs, etc.) to be integrated on a single chip. These are often referred to as Systems-on-Chip (SoCs) [1].

SoCs have been widely recognized as a possible and cost-effective means for achieving performance beyond that achievable from a single processor [1]. In such systems, concurrent tasks inherent in applications are distributed over a group of processing elements to run simultaneously. These processing elements are connected by an interconnection network, and exchange information in their activities to solve a common problem. In order to permit processing elements to focus on computation and allow the overlap of computation and communication, each processing element is associated with a routing element that is responsible for handling message communication. The assembly of the processing and routing elements is called a *node* [2].

The performance of SoCs can be affected by considerations of different levels, ranging from the way application processes are distributed among the various processing elements, to the efficiency of the underlying interconnection network. The latter has been the focus of much recent research activities as any interaction between the processing elements highly depends on the efficiency of the underlying network, often referred to as a Network-on-Chip (NoC).

The performance of NoCs is usually measured in terms of *response time* and *throughput*. The first metric represents the speed of the network, and is the time taken for a message to cross the network. Throughput, on the other hand, is the number of delivered messages in the network per unit of time, and represents the load handling capacity of the network. Ideally, an NoC should provide low response times and high throughput. However, there are many factors that can affect NoC performance of which the most important are, *topology*, *switching*, *routing* and *implementation constraints* [2].

The manner in which nodes are connected via communication links in an NoC is referred to as a *topology*. Widely used topologies for NoCs include the mesh, torus and fat-tree topologies [1]. The properties of NoC topologies have been extensively researched over the past few decades. Most of these studies have focused on the graph-theoretical properties of the NoC topologies and have ignored the constraint imposed by implementation technology. For NoCs where the whole system is implemented on a single chip, the most relevant constraint is the wiring density used by a given topology. Wiring density describes the number of wires required by the topology. This wiring density is usually fixed and limited by the implementation technology. Dally [3] has introduced the *bisection width* as a means for quantifying the wiring density requirement of a given topology when laid out on a 2D VLSI chip. The bisection width is the number of wires that cross the center of the chip. Assuming a fixed wiring density, a topology with more links crossing the center ends up with less wires per link, and as a consequence, lower channel width (bandwidth) for each link.

The purpose of our project is to revisit the relative performance merit of well-known topologies for NoCs including the mesh and the torus when implementation constraints are taken into account. In order to achieve this, a software simulator has been developed for the well-known topologies notably the mesh and the torus using the discrete–event simulation technique. The simulation models are then used to carry out an extensive comparison among such topologies for both unconstrained implementations as well as constrained implementations.

Outline of the report

Chapter 1 provides a technical background on NoCs and the important factors that affect their performance, including topology, switching and routing.

Chapter 2 provides an overview on the existing the related research work which has compared the performance of various NoC topologies along with a critical summary of these studies and also gives justification for adopting the simulation approach in order to conduct our study, then presents the system model that has been used for developing the simulator.

Chapter 3 describes the simulation model (in pseudocode) using the discrete-event simulation technique.

Chapter 4 uses the simulation model in order to conduct extensive comparison between the well-known NoC topologies under various operating conditions and discusses the obtained performance results.

Chapter 1: Background on networks-on-chip

The performance of NoCs can be affected by several factors including *topology*, *switching* and *routing* [2]. This chapter provides an overview of these factors. Our aim is to provide the necessary technical background required for understanding the subsequent chapters in this project report.

1.1 Network-on-Chip topologies

A topology of an NoC is typically modeled as a graph G=(V, E), where V, the set of vertices, represent the nodes, which contain the processing and routing elements, and E, the set of edges, represent the communication links interconnecting the nodes.

The topology depicts how nodes are connected with each other via communication links. A topology can be "logical" and/or "physical". The logical topology illustrates how data flows among the nodes within the network whereas the physical topology indicates the placement of the various nodes on the chip area.

Existing topologies for NoCs can be classified into *direct* or *indirect*, and can also be classified as *regular* and *irregular* [1]. The choice of a given topology for an SoC often depends on the application's communication requirements.

A typical node is composed of a processing element that can include CPUs, GPUs, DSPs, etc., local memory, and a router with input and output links. A generic node architecture is depicted in Figure 1.1, and will be used in the subsequent sections in this work.

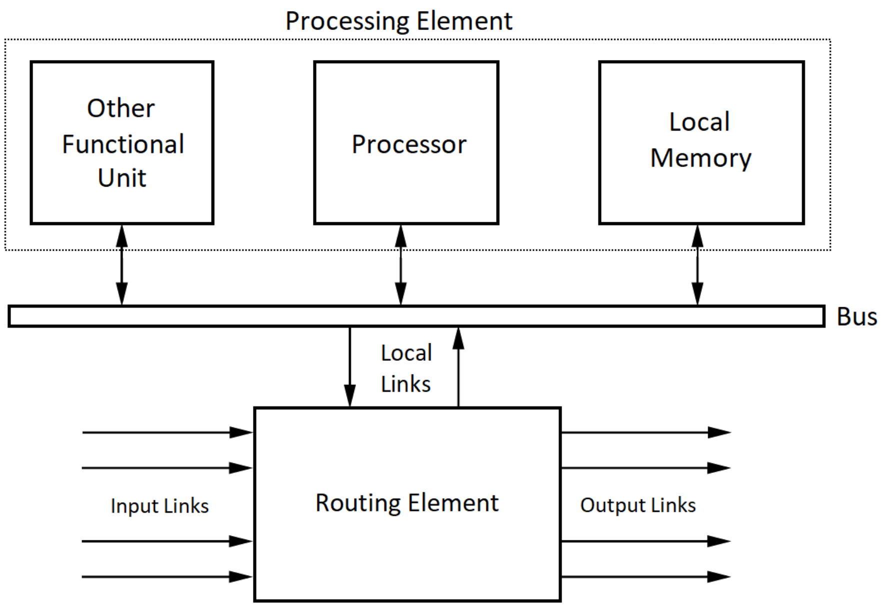


Figure 1.1 An example of a generic node architecture for NoCs.

1.1.1 Direct and indirect topologies

In direct topologies, each node is directly connected to a subset of other nodes in the network. Nodes in a direct topology contain both processing and routing elements. This class of topologies is known for its high scalability [2]. Examples of well-known direct topologies include *meshes*, *tori*, *spidergon* and *trees* [3].

On the other hand, nodes in indirect topologies separate the processing elements and routing elements (which are referred to as switches [4]). For processing elements to communicate with each other they must pass through the switches. Each processing element has a network adapter which allows it to connect to a switch. A switch has a set of ports, each having an input and output link. Ports are used to connect with processing elements or other switches. The interconnections of these switches define the various topologies. Indirect topologies are useful when specific patterns are required for specific applications, for example image/video processing. Examples of well-known indirect topologies include butterflies, deBruin, and Clos networks [2]. Figure 1.2(a) and Figure 1.2(b) depict an example of a direct and indirect topology respectively.

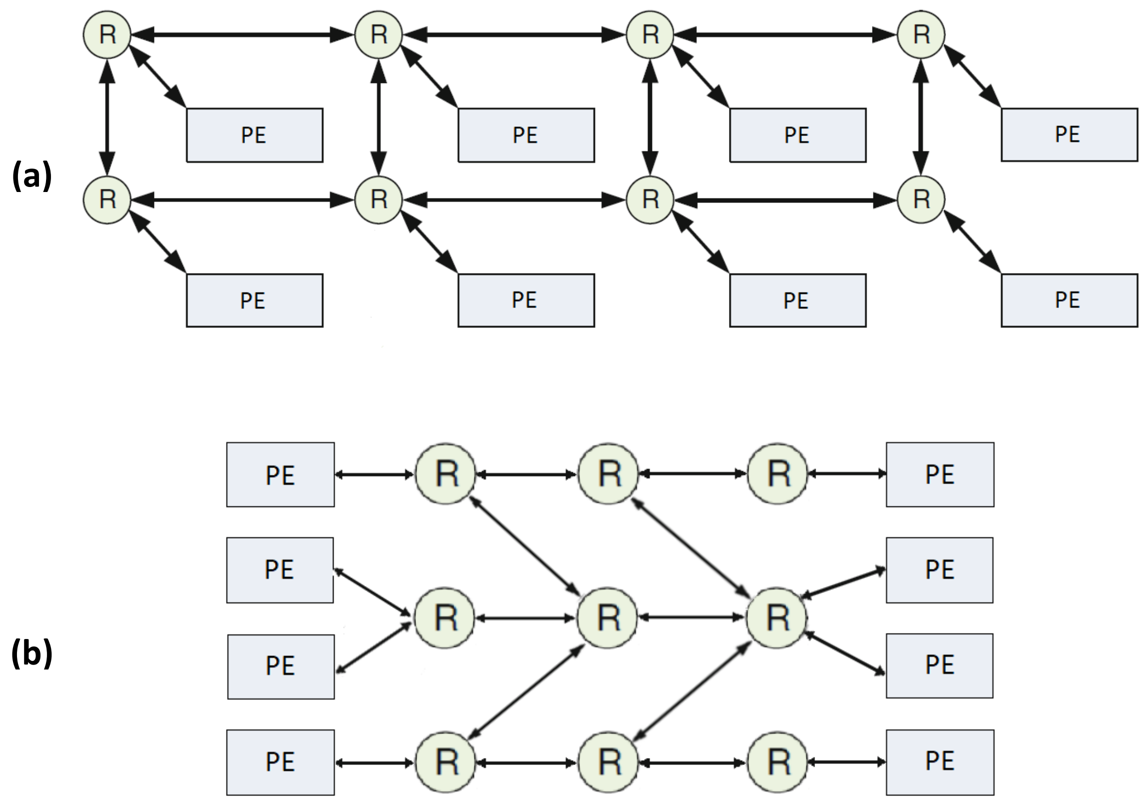


Figure 1.2 An example of: (a) Direct and (b) Indirect topology [1].

1.1.2 Regular and irregular topologies

Each node in regular NoC topologies has a comparable number of neighboring nodes. They are simple to implement since they are not designed for any specific application. This also comes with a drawback which is that these topologies are not optimized for specific applications [5]. On the other hand, nodes in irregular topologies have highly varying number of neighbors. This can allow them to be dedicated for specific applications, but are more difficult to implement in practice.

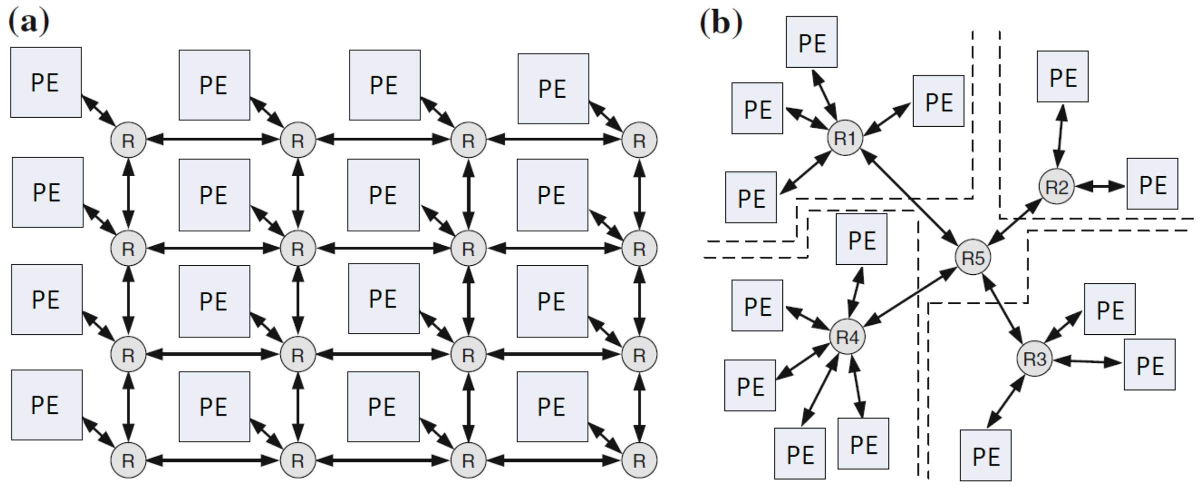


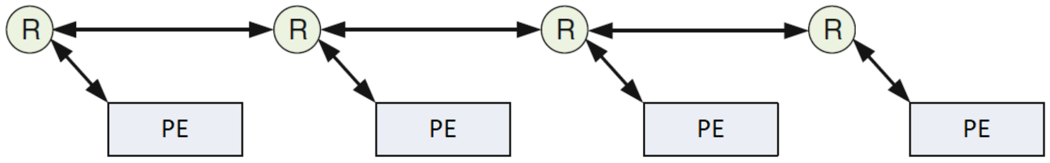
Figure 1.3 An example of: (a) Regular and (b) Irregular topology [1].

Figure 1.3(a) and Figure 1.3(b) depict an example of a regular and irregular topology respectively. What follows is a brief description of well-known topologies used in NoCs.

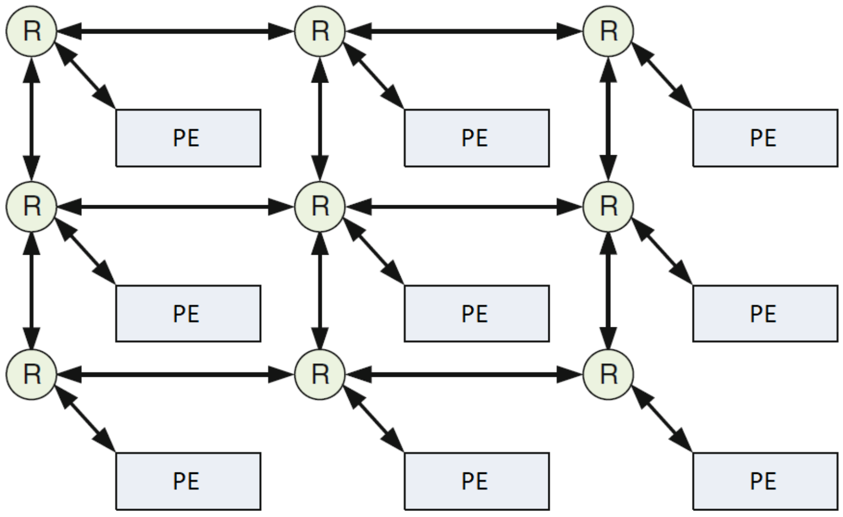
1.1.3 N-dimensional mesh

The n-dimensional mesh is one of the most widely used topologies. It is *strictly orthogonal* [6]; a topology is orthogonal if and only if nodes can be arranged in an orthogonal *n*-dimensional space, and every link can be arranged in such a way that it produces a displacement in a single dimension. Strictly orthogonal means that every node has at least one link per dimension. The advantage of holding this property is the ease of implementing routing algorithms which makes this topology one of the easiest to implement [6]. Below are the figures of the 1D mesh, 2D mesh and 3D mesh. However, the mesh does suffer from some drawbacks including its large network diameter (i.e. the longest distance between two nodes) and the accumulation of traffic towards the center because the mesh topology is asymmetric. Figure 1.4(a), 1.4(b) and 1.4(c) depict the 1D, 2D and 3D mesh respectively.

(a)



(b)



(c)

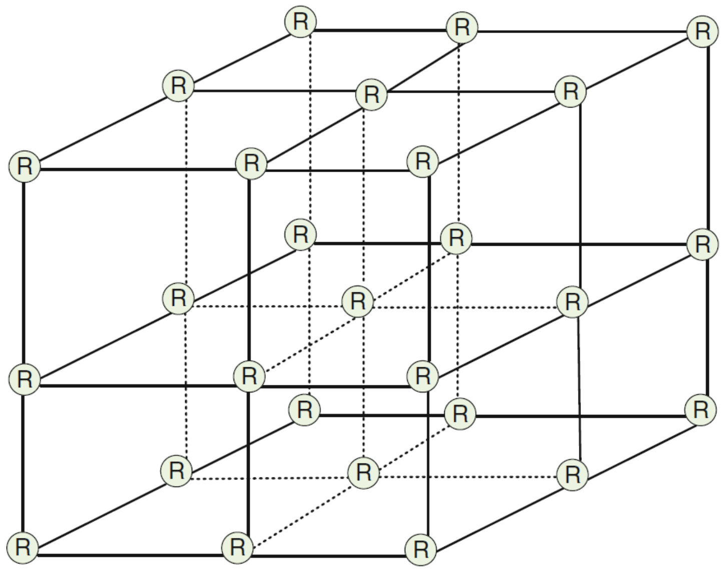
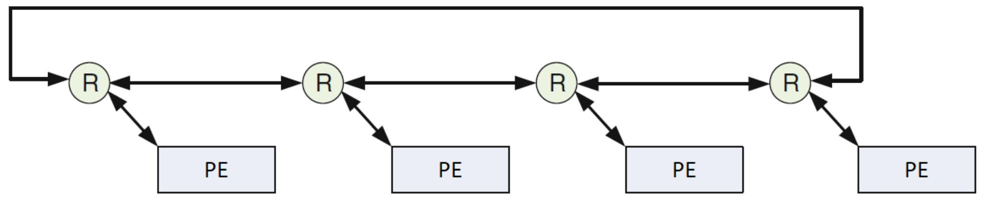


Figure 1.4 An example of: (a) 1D Mesh, (b) 2D Mesh and (c) 3D Mesh [1].

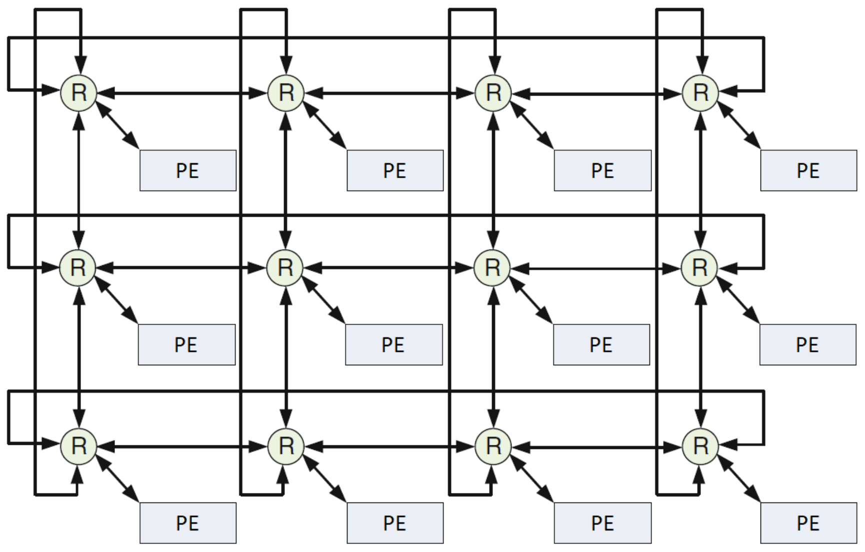
1.1.4 N-dimensional torus

The torus is another strictly orthogonal topology that is similar to the mesh. However, the ends of the rows and columns are connected with each other. The links can be unidirectional or bidirectional. This is a workaround for one of the shortcomings of the mesh which is its large diameter and its unbalanced traffic. Thus, it provides advantages over the mesh as it has a lower diameter that reduces latency and makes the topology symmetric which reduces traffic pressure in the center. Such benefits however come at the cost of increased wiring density and the need for more complex routing algorithms due to its long wraparound links [7]. Figures 1.5(a), 1.5(b) and 1.5(c) show the 1D, 2D and 3D torus.

(a)



(b)



(c)

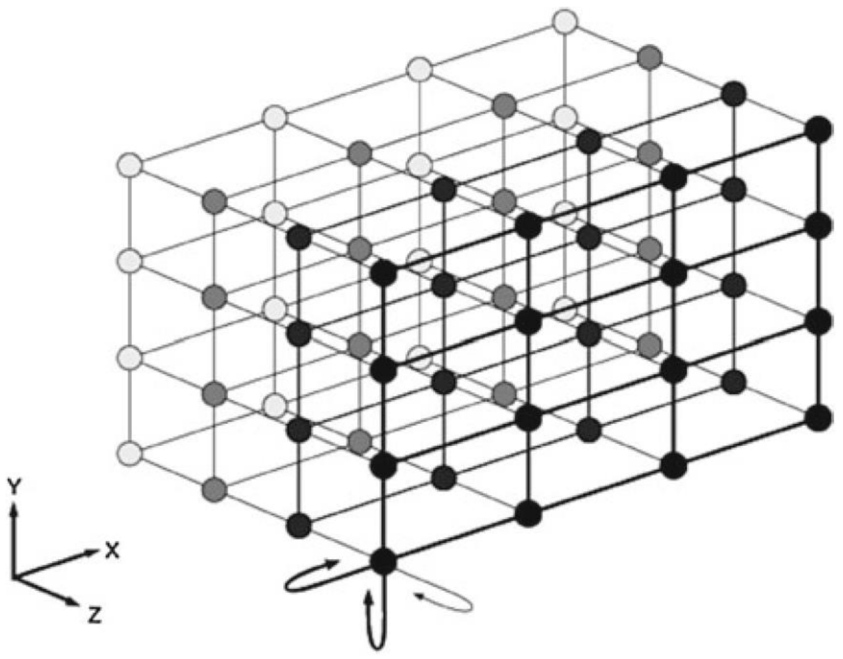


Figure 1.5 An example of: (a) 1DTorus, (b) 2D Torus and (c) 3D Torus [1].

Folded torus: This is a proposed variation of the torus to remedy the problem of its long wrap-around links. In this variation, the links are folded in order to create the same physical length for all links between nodes and this is very useful for practical implementations. However, this comes at the expense of requiring a larger surface area on the chip and an increased amount of links crossing the center of the chip [8]. Figure 1.6 shows the folded torus.

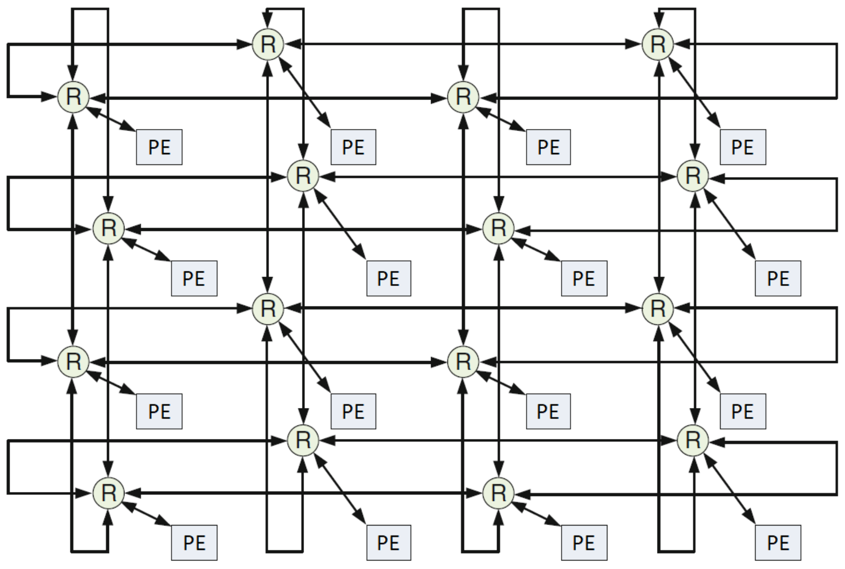


Figure 1.6 An example of folded torus [1].

1.1.5 Trees

Many topologies are structured around trees. A tree [9] is composed of a root node connected to a disjoint set of descendants. A node without descendants is called a leaf node. An interesting property of trees is that every node except the root has only one parent, this means that trees do not allow for cycles, making it easier to avoid detrimental issues such as deadlocks. However, one of the main drawbacks of trees is that the root node and its closest descendants can form a bottleneck as most of the traffic tends to accumulate there. Figure 1.7 depicts an example of a tree.

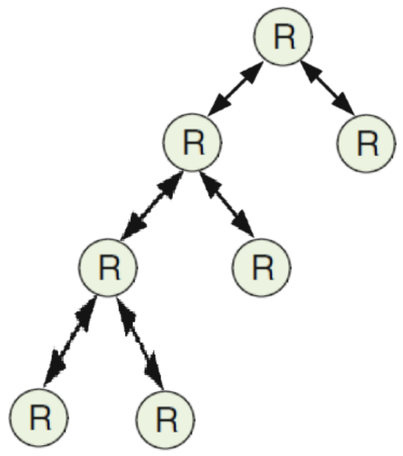


Figure 1.7 An example of a tree topology [1].

Fat-tree: is one of the common tree topologies adopted in NoC architectures [10]. It attempts to work around the core issue of bottleneck near the root by using gradually higher bandwidth as the links get closer to the root node, with the highest bandwidth links starting directly from the root node. As seen in the figure below, the links represented with a larger number of arrows represent higher bandwidth. Figure 1.8 illustrates an example of the fat-tree.

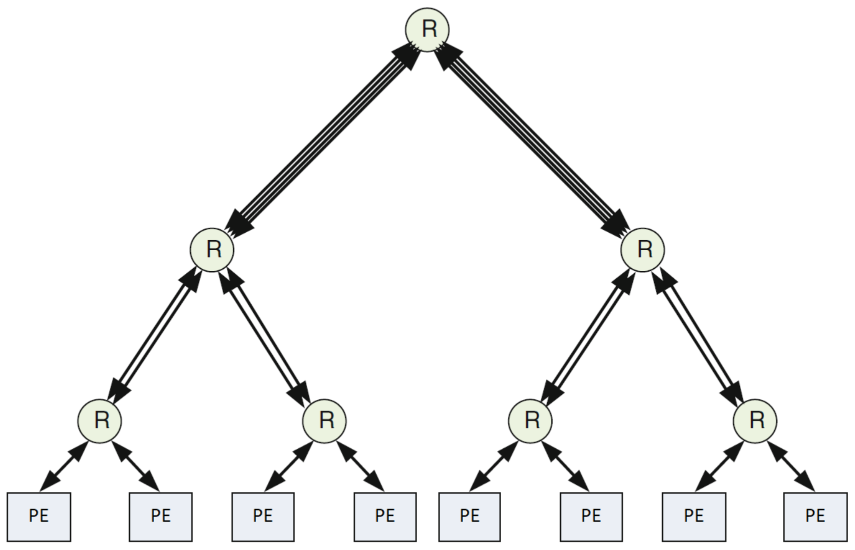


Figure 1.8 An example of a fat tree topology [1].

1.1.6 Butterfly

The butterfly [11] is an example of an indirect topology. It has a simple recursive structure that takes full advantage of its high number of routes and reduced network latency overhead which produces overall good performance. The links can be either uni or bidirectional and generally consist of input and output ports and router stages that contain routers. Each packet that arrives to the input of a router is directed (routed) to the proper output. Figure 1.9 shows an example of the butterfly.

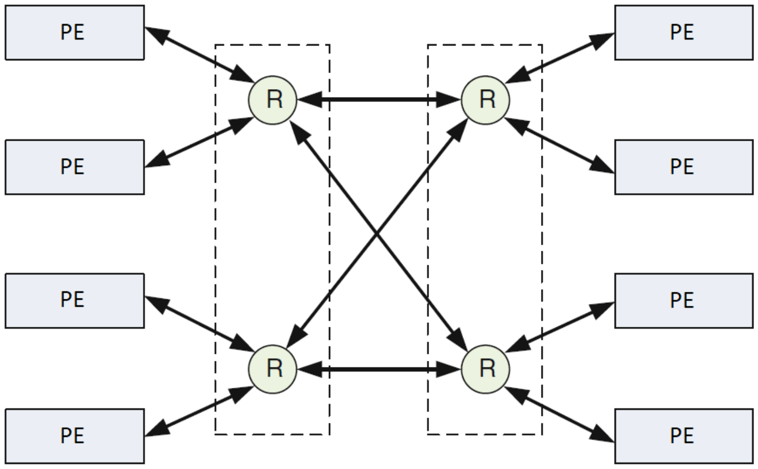


Figure 1.9 An example of a butterfly with 4 input ports, 4 output ports and 2 router stages each contains 2 routers [1].

1.2 Topological properties

In this work, we will focus on direct and regular topologies because they have been widely used for NoCs compared to the other classes of topologies. Direct regular topologies have various important characteristics such as *degree*, *diameter, average distance* and *bisection width*. Such characteristics are often used to analyse the difference in performance among competing topologies in comparative evaluation studies [12].

1.2.1 Node degree

It is the maximum number of neighboring nodes that are connected to a given node. A network is only called regular if all nodes have the same fixed degree; otherwise, it is irregular.

The node degree can be constant or varies as the network scales up. Higher node degree reduces the average distance whilst a smaller node degree reduces hardware implementation costs. This creates a constraint on node degree when it comes to implementing NoC topologies [12, 13].

1.2.2 Diameter

The diameter of a network is the maximum shortest path between any two nodes [12, 13]. If there is no direct connection between two nodes, a message has to travel through intermediate nodes which introduces hop delay. Since the message delay is proportional to the number of hops, the length of the maximum shortest path becomes an important factor in determining network performance. A small network diameter can provide predictable routing paths and traffic flow and thus low latency.

1.2.3 Average distance

The average distance provides an indication of the average number of hops that exist between any given pair of nodes in the topology. This also gives an indication on the delay that a packet experiences when the traffic is uniformly distributed (i.e. a message is equally likely to be destined to any other node in the network).

1.2.4 Bisection width

Bisection width is an important characteristic that reflects the number of links that cross the center of the topology when it is laid out on a plane. A large bisection width is preferable, because it provides more paths between the two halves of the network and thus improves overall performance. Table 1.1 depicts a comparison of the mesh, unidirectional and bidirectional torus based on several graph-theoretic properties.

Table 1.1 Comparison of the topological properties of the mesh, unidirectional and bidirectional torus with $N = k^n$ nodes where *k* is the number of nodes per dimensions and *n* is the number of dimensions (please see [2] for more details).

Network	Degree	Diameter	Average distance	Bisection width
n-dimensional mesh	2*n*	*n*(*k*-1)	$\frac{n}{3}(k - \frac{1}{k})$	$2\sqrt{N}k^{\frac{n}{2}-1}$
n-dimensional bidirectional torus	2*n*	$n\frac{k-1}{2}$	$n\frac{k-1}{4}$	$\frac{4N}{k}$
n-dimensional unidirectional torus	*n*	$n(k-1)$	$n\frac{k-1}{2}$	$\frac{2N}{k}$

1.3 Switching

Switching determines how packets are transferred between different routing elements and how they are buffered. Data can be divided into several categories depending on size [1]:

- Messages: Group of packets that makes the entire data.
- Packets: Group of sequential flits with the same destination.
- Flow Control Units (flits): unit of synchronization between routers.
- Phits (physical units): unit of data transferred through the physical link in a given cycle.

Listed below are brief descriptions of the more common switching techniques.

Circuit switching:

In circuit switching [14], a physical link is established between source and destination prior to data transmission. The routing header is injected into the network. The header contains the destination address and is called the routing probe. The routing probe progresses through the network to the destination reserving physical links as it is transmitted through the routers. Once it arrives at the destination, the connection is fully established and an acknowledgement is sent back to the source. The full bandwidth supported by the links is now available for data transmission.

Circuit switching is most advantageous when messages are large in size and with large time periods in between consecutive message arrivals at routers. However, a major downside of this technique is that the entire physical path is blocked during transmission which can block other messages.

Packet switching:

In packet switching, a message is split into packets. The first few bytes of a packet contain routing and control information and is called the packet header. Each packet is individually routed from the source to the destination. A packet is stored in a buffer in each intermediate node before being transmitted to the next node. This is the reason this technique is also called Store-and-Forward [15].

In contrast to circuit switching, this technique is at its most advantageous when messages are short and frequent. Also contrary to circuit switching, links are fully utilized as multiple packets belonging to different messages can be in the network simultaneously and no physical links are blocked. However, the buffering and packet assembly/de-assembly can lead to overhead and a reduction in overall throughput.

Virtual cut-through:

Packet switching assumes that a packet must be received and buffered at a node in its entirety before any decisions can be made on where to forward it [16]. Given that the first few bytes of a packet contain routing information, virtual cut-through allows a packet to be forwarded to the next node whilst it has not even been fully received at the current node. The message can be effectively pipelined to its destination in low and moderate traffic loads.

Assuming there is no blocking the latency experienced by the header at each node is the routing latency through the node and propagation delay along the physical channels. If the header is blocked on a busy output channel, the entire message is buffered in that node. Under high network loads, virtual cut-through behaves increasingly similar to packet switching.

Wormhole:

The need to buffer entire packets can make it expensive and difficult to create fast and small routing elements. Similar to virtual cut-through, in wormhole switching [17] messages are also pipelined across the network. However, the major difference compared to the above-mentioned techniques is the largely reduced buffer size requirement at the nodes. A packet is broken into flits. The flit is the unit of message flow control, and input and output buffers at a node are typically large enough to store a few flits. The message is pipelined through the network at the flit level whilst the message is typically too large to store in a node.

At any given time, a message can be occupying several different nodes at once. The primary difference between this and virtual cut-through is that, in the former, the unit of message flow control is a single flit and, as a consequence, smaller buffers can be used. Just a few flits need to be buffered at a node. The major drawback of this technique is the fact that a single message can be spread out across multiple nodes and thus occupying multiple buffers which may lead to blocking and deadlock problems [2].

Table 1.2 Comparison of the switching techniques.

Switching	**Performance properties**	**Design complexity**	**Buffering**	**Cost**	**Adaptability to traffic**
Circuit switching	Good under light traffic	Low	1 flit	Low	None
Packet switching	Good under moderate/heavy traffic	Low	Packet	High	High
Virtual cut-through	Good under light/moderate/heavy traffic	High	Packet	High	High
Wormhole	Good under light traffic	Low	A few flits	Low	Low

1.4 Routing

Routing algorithms determine the path followed by each message or packet between source and destination [18]. Properties that are desirable from routing algorithms include:

- Connectivity: Ability for a packet to be sent from any source to any destination.
- Adaptivity: Ability to find alternative paths in case of faulty components or congestion.
- Guaranteed to be deadlock and livelock free: These issues can prove detrimental to a network and thus routing algorithms must be guaranteed to be free of these.

Routing can be categorized into three classes: *Deterministic*, *adaptive* and *stochastic* [6, 20, 21].

Deterministic routing:

This creates a path as a function of the destination address. This means that between any two nodes the path created will always be the same regardless of the state of the network. The main advantage of this type of routing is the simplicity of the design of routing elements which creates a low latency when traffic is low. The drawback however, is lower flexibility in dealing with changing traffic conditions.

Adaptive routing:

Adaptive routing takes into consideration the state of the network before making any routing decisions. This leads to increased flexibility at the cost of more difficult implementation due to higher routing complexity.

Stochastic routing:

Routing decisions are made without knowledge on the state of the network. Whilst at first this might sound similar to deterministic routing which also does not take the state of the network into account, the main difference is that deterministic routing always takes the same choices. Stochastic routing may use a different manner (randomly or cyclic) to make its choices which is independent of the state of the network.

More on routing:

Routing algorithms can also be *minimal* or *non-minimal* [21]. Minimal algorithms always choose the shortest path between any two nodes. This leads to reduced latency however avoiding deadlock can prove to be a challenge.

Non-minimal routing allows a message to move away from its destination. One of the reasons for this could be to avoid creation of cycles and deadlocked configurations however the major downside is the threat of livelock.

Another technique associated with routing calculation is the distinction between source routing and distributed routing [1].

- **Source:** The source node calculates the proper path and stores it in the packet header, since the header must be forwarded through the network to reach the destination. The intermediate nodes do not make any routing judgments thus allowing for simpler routing elements.
- **Distributed:** The packet header only contains the destination address which makes the routing path determined by each node on its way to the destination. Distributed routing can be adaptive to network changes.

Deadlock and livelock [2] are fundamental problems that can appear in networks and consequently routing algorithms must be able to effectively deal with them.

In a typical NoC, buffers are used to store packets or fragments of packets. Since buffers offer a limited size a situation can occur where packets cannot progress to their destination because the buffers, they need to go through are fully occupied. Meanwhile those buffers are occupied by packets which cannot progress either since the buffers they are requesting are also occupied. Packets in a deadlocked configuration end up being permanently blocked. This is why avoiding deadlocks is extremely important for a routing algorithm otherwise packets would end up never reaching their destination. A simple solution for deadlock is to use deterministic routing [2] or drop packets similar to what is done currently in the internet.

Livelock occurs when packets continue to move through the network and spin around its destination without ever reaching it. Usually appears when non-minimal routing algorithms are used. A simple solution to avoid livelock is to use minimal routing (such as deterministic) or dropping packets.

1.5 Conclusions

This chapter has presented the most critical factors that affect the performance of NoCs. These include topology, switching and routing. We have presented the topological properties of NoCs and described some well-known topologies including the mesh and torus. We have also described several switching techniques including packet switching, circuit switching, virtual cut-through and wormhole switching. After that we described the routing techniques commonly used in NoCs including deterministic and adaptive routing.

The next chapter will present the related research work on the performance evaluation of some well-known NoC topologies which have been reported in the literature.

Chapter 2: Related research work and simulation modeling

2.1 Related research work

This section surveys a number of existing comparative studies that have been carried out on various topologies for NoCs either using analytical modeling or simulation. Our aim is to provide an updated review of the research carried out in this area.

2.1.1 2D Mesh vs. 2D Torus

The simplicity and popularity of the 2D mesh and 2D torus has made them a common choice for comparisons. In [22], the metrics used to determine the overall performance of the 2D mesh and 2D torus are throughput, latency, and power consumption. The study used as parameters the hop count, bisection width and wire length for interconnections which in turn are decisive factors for determining power consumption/dissipation. The comparison has been performed analytically using the basic concepts of graph theory and the results have been validated through the simulation of the two topologies. The performance results reported in [22] indicate that as the network size increases, the performance of the 2D mesh worsens considerably compared to the 2D torus. However, it can be noted that the area and power requirements are higher in the torus due to the increase in wire length in practical implementations.

In [23], a different analysis of the two topologies has been carried out using software simulation. The metrics used in this evaluation were latency, power consumption and power-throughput ratio. The evaluation was performed using deterministic, partially adaptive, and fully adaptive routing whilst the traffic was modeled using uniform and hotspot distributions. Similar to the analytical comparison of [22], the simulation results have shown that the torus has reduced latency compared to the mesh, whilst exhibiting higher power consumption.

2.1.2 2D Mesh vs. Fat-tree vs. Butterfly fat-tree

In the work of [24], three area-efficient topologies have been compared, and these were the 2D mesh, fat-tree (FT) and butterfly fat-tree (BFT). The OPNET simulator was used which provides a convenient environment for hierarchical modeling of networks. It was assumed that the networks use wormhole and virtual cut-through switching.

The metrics examined in this evaluation were latency and throughput. A finite buffer size was assumed and a uniform traffic pattern with different injection rates was used to obtain different communication scenarios. The results have revealed that FT exhibits the lowest latency and the highest throughput. This is due to its higher bandwidth links near the root. Even though BFT has lower throughput and higher latency, the lower number of routing elements and links lead to a lower area overhead and energy dissipation. The results have also indicated that virtual cut-through outperforms wormhole in all three topologies. The main conclusion from this analysis is that using FT in tandem with virtual cut-through could be an effective solution for NoC design.

2.1.3 2D Mesh vs. Torus vs. Spidergon

Whilst most existing comparisons have been conducted on regular topologies, one of the first studies involving irregular topologies such as the spidergon [25] against the 1D torus and 2D mesh [26]. The simulation modeling of the NoC architectures was performed using the OMNeT++ simulation framework. Figure 2.1 shows an example of the spidergon and 1D torus.

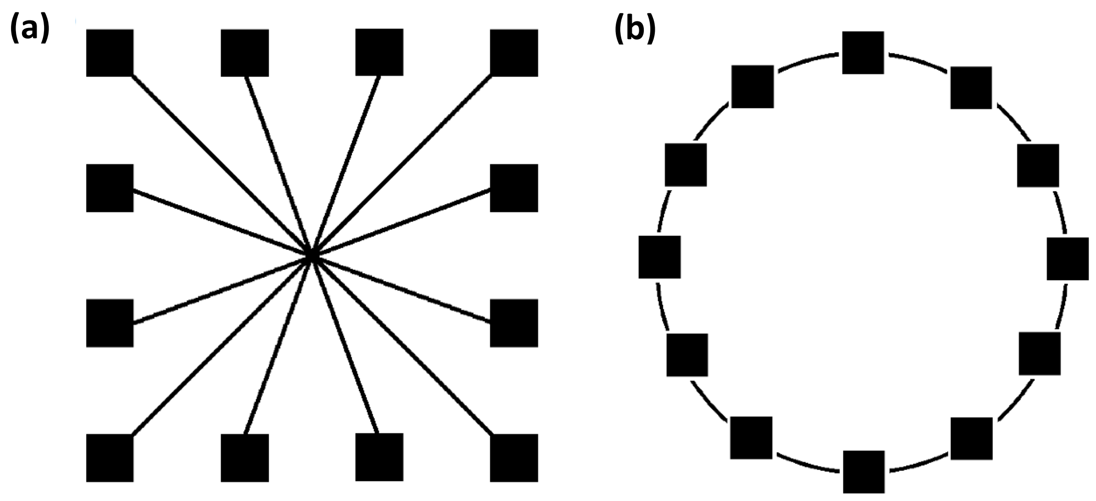


Figure 2.1 An example of: (a) Spidergon and (b) Ring Topologies

Packet inter-arrivals at a given node follow a Poisson distribution. Packets are of fixed length. The evaluation assumed limited buffer sizes and wormhole switching was used in all three topologies. The metrics considered were NoC throughput and latency. The topologies taken into consideration were put to the test under uniform workloads.

Results were collected for three traffic scenarios: single hotspot, double hotspot and uniform. The single hotspot scenario has one node acting as the destination node for all messages. The double hotspot has two nodes act as destinations whilst the uniform has all nodes have an equal probability of being the destination.

The conclusion drawn from the results of the three traffic scenarios is that the spidergon, given its ease of implementation could be an attractive solution as it delivers performance results and scalability comparable to that of more complex solutions.

2.1.4 2D Mesh vs. Square-octagon

In the study of [27], the square-octagon was introduced and compared against the 2D mesh.

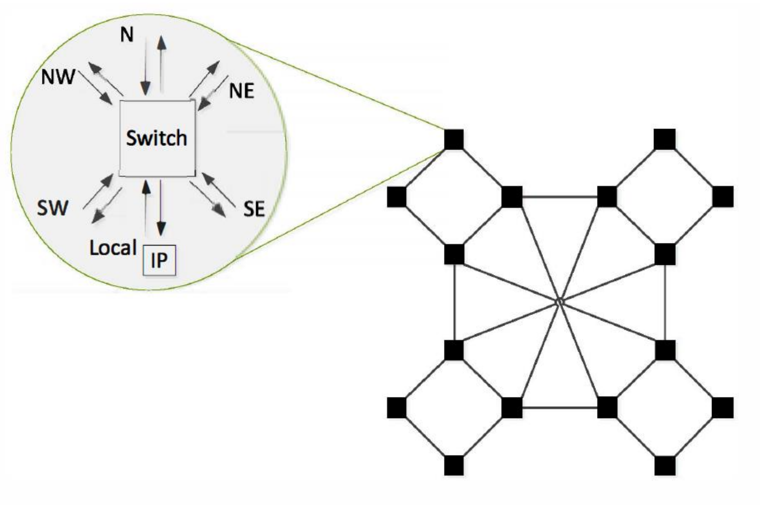


Figure 2.2 An example of the Square-Octagon

Figure 2.2 shows a basic module of the square-octagon (SONoC) with 16 nodes which are connected using 24 bidirectional links. SONoC is built by using 4 squares and one octagon to connect them using diagonal links in between. Each square is considered a cluster, and each cluster contains four nodes.

The performance comparison between the SONoC and 2D mesh was carried out by using the OPNET simulator software. The considered metrics were throughput and latency. The topologies were tested under different traffic patterns such as uniform and hotspot. The considered parameters were the degree of the network, diameter, average hop count, path diversity, number of links and bisection width.

The comparison shows that the SONoC displays better results than the mesh for different network sizes as SONoC performs better as the network size increases. Final results show that the SONoC outperforms the 2D mesh in terms of latency and throughput due to the octagon clusters that provide diagonal links which affect the diameter of the topology as well as the degree which provides rich path diversity.

2.1.5 2D mesh vs. Honeycomb

The authors in [28] have introduced the honeycomb as an alternative to the mesh and torus. Also known as honeycomb mesh, it is composed of a number of hexagons as indicated in Figure 2.3. The comparison was carried out against the 2D mesh.

The simulation was performed by using a simulator called Orion 2. The used metrics are power consumption, area cost in addition to latency. The considered parameters are the network diameter and the node degree.

The results show that the honeycomb has better overall performance than that of the mesh; the honeycomb has a lower network cost, consumes less power and saves more area than the mesh.

It was also found that the communication delay is reduced compared to the mesh which makes the honeycomb a preferable choice for NoC architectures.

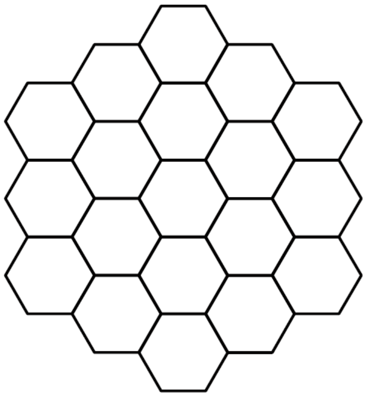


Figure 2.3 An example of the honeycomb mesh.

Table 2.1 provides a summary of the existing works mentioned above.

Table 2.1: Summary of related work

Authors	Topologies	Metrics	Parameters	Findings
V. Sanju, Niranjan Chiplunkar, M. Khalid, Sujata Josh and J. S. Nirmala (2013)	**2D mesh vs 2D torus**	Throughput, latency, and power consumption	Maximum hop count, average hop count, number of wires and wire length	As the size of the mesh increases, the maximum hop count and the average hop count increases, which leads to worse performance. In torus' case the performance does not worsen but the increase in wire length and quantity lead to higher power consumption
M. Mirza Aghatabar, S.Koohi, S.	**2D mesh vs 2D torus**	Latency, Power consumption and	Routing algorithms, traffic model	The torus has better latency than the mesh at cost of higher power consumption. For latency the torus is the

Hessabi and M. Pedram (2007)		Power/Throughput ratio	and number of virtual channels	better option and for power consumption the mesh is a superior choice
Wu Ning, Ge Fen and Wang Qi (2007)	**2D mesh vs fat-tree vs butterfly fat-tree**	Throughput, latency	Wormhole and virtual cut-through switching techniques	Fat-tree shows the best results with the lowest latency and highest throughput out of the three topologies.
Luciano Bononi and Nicola Concer	**2D mesh vs ring vs spidergon**	Throughput, latency	Wormhole	Out of the three topologies, spidergon has displayed best results, and as an irregular topology, spidergon appear to have trade-off solution for getting same performance as the complex architectures
Meaad Fadhel Ali Qasem and Huaxi Gu (2014)	**2D mesh vs square octagon**	Throughput and end-to-end delay	Degree, diameter, average hop count and path diversity	The square-octagon has higher throughput and lower end-to-end delay than the 2D mesh
Alexander Yin, Nan Chen, Pasi Liljeberg and Hannu Tenhunen (2011)	**2D mesh vs honeycomb**	Power consumption, area cost and communication delay	Degree and diameter	As an alternative implementation of NoC based systems, the honeycomb displayed better performance than the 2D mesh. It exhibited lower communication delay, less power consumption and almost half the area cost

2.1.6 Summary

In this section we have reviewed some research studies that have compared the performance of some well-known topologies for NoCs. These comparisons include mesh versus torus, mesh versus fat-tree and butterfly fat-tree and mesh versus ring versus Spidergon. However, these comparisons have based on the topological properties but have not taken into account the constraints imposed by implementation technology such as the wiring density. This constraint can severely limit the bandwidth of channels in a given topology which may greatly impact network performance including message delay and throughput.

The aim of our study is to convincingly show that implementation constraints have to be taken into account when comparing the relative merit of NoC topologies as they may greatly impact the outcome of any comparative study.

The next section presents the different types of simulations and discusses in detail the discrete-event simulation technique which has been adopted in our study.

2.2 Simulation modeling

This section starts off with a justification as to why simulation has been adopted in our study, followed by a presentation of various simulation techniques. We then discuss the discrete-event simulation technique in detail. After that we present the system model that we used in our simulation.

2.2.1 Justification of the method of study

In order to perform the performance comparison between the competing topologies, software simulation has been selected. Simulation was chosen over the analytical approach because the analytical models often resort to simplifying assumptions and ignore many system details which results in reduced prediction accuracy [29]. Moreover, some studies [22] have analyzed the static properties of NoC topologies using for instance graph theory. However, such studies do not consider time dependent behavior of the system which may not be captured by the *static* analysis. Furthermore, it is a complex undertaking to capture analytically the dependencies between system parameters when determining system performance. A real-life implementation of the system is not an option in our case due to lack of funding and computing resources.

Simulation has been used to conduct our study as it provides a good trade-off between implementation cost and accuracy of prediction. Various NoC simulation environments exist (including OPNET [24, 27], OMNeT++ [25], Orion 2 [28]). However, these simulators are either proprietary, or not widely used by the research community on NoCs. This has made resources on how to operate these simulators scarce. Moreover, these simulators often contain unnecessary details which are irrelevant to our present study. Besides, adapting existing simulators for the purpose of our comparisons may prove time consuming and a challenging task. Consequently, it has been decided to develop our own simulator from the ground up using the discrete event simulation technique [30].

In what follows, we will briefly review the different types of simulations. Then we will present the system model of a node in an NoC.

2.2.2 Types of simulation models

Simulations are useful because they allow prediction of how systems operate without having to implement them in the real world. They also allow prediction of various possible failures in the system design due to the impact of different modifications to the system.

Monte-carlo simulation:

Monte-Carlo models probabilistic phenomenon that do not change over time [31]. This type of simulation is inherently static (i.e. it is assumed that time is fixed). It utilizes statistical tools to mathematically model a real-life system or process and then estimates the probability of obtaining a successful outcome. It entails using random numbers as a tool to compute a function that is not random. Possible applications of this technique include the simulation of random and stochastic processes (such as traffic flow) and evaluation of integrals. However, this technique is not suitable for our research study as it cannot model the system behavior over time.

Trace-driven simulation:

This type refers to system simulations performed by looking at traces of program execution or system component access with the purpose of predicting performance. Trace-driven simulation [33] uses time-ordered records of events on real systems as an input. It usually has two components: one that executes actions and stores the results and another which reads the log files of traces and inserts them into new scenarios. Possible applications include scheduling, caches and analysis of solid-state disks.

Discrete event simulation:

Discrete event simulation is a technique which can be used on systems that can be represented by a queuing model. The purpose of discrete event simulation is to analyse the behavior of the system over time. The system is characterized by a group of *state variables* and by operators that manipulate these variables [30, 32].

In discrete event simulation the system is only studied when a change in state occurs. This is usually a result of an *event* taking place. The simulator examines the system at discrete time intervals, processes any events that might have occurred and changes the state variables accordingly. New events are then generated as a result of transitioning to the new state.

The events are stored in a queue known as the *event list* where each item in the queue contains the time of the event, the type of the event and the location where it occurs. The events in the queue are sorted in order of when they occur, with sooner events having a higher priority in the queue. There are generally two types of events: *primary* events, events that do not depend on any event other than time, and *conditional* events which are triggered as a result of other events. Each event refers to a procedure which is executed when that event is processed. Table 3.1 shows a comparison between the different classes of software simulation.

Table 2.2: Comparison between simulation models

	Monte-Carlo simulation	**Discrete-event simulation**	**Trace-driven simulation**
Advantages	Using MC simulation is straightforward.	Allows study and experimentation with a complex system.	Less randomness - deterministic input reduces output randomness.
	Provides approximate solutions to many mathematical problems.	Enables the feasibility testing of any hypothesis about how or why certain phenomena occur.	Detailed tradeoffs - possible to evaluate small changes in model.
	Provides statistical sampling for numerical experiments using a computer.	Evaluates the different circumstances of simulation by changing the inputs and observing the resultant outputs.	Easy validation
Disadvantages	The results are only an approximation of the true value.		Complexity - requires detailed simulation of system
	Simulation results can show large variance.		High level of detail - simulations can be costly.
	A single sample cannot be used is simulation; many samples are required to obtain results.		Hard to evaluate changes in workload characteristics - need another trace.

2.2.3 System model

For the purpose of our study, the NoCs are modeled as a set of nodes connected with links. Each node is given a designated address which consists of *n* components, with *n* being the number of dimensions in the topology. In each dimension, there are *k* nodes, and therefore the network size is $N=k^n$. For example, in the case of a 2D topology the address for any given node is designated as (*x*,*y*) with $0 \le x,y < k$. In a 3D topology the address for any given node is (*x*,*y*,*z*) with $0 \le x,y,z < k$. For the sake of brevity, we will describe the node structure for the 2D mesh only. This is because the description also applies to the node structure in the other topologies (such as the torus) and higher dimensions with only minor modifications.

Node model:

In the 2D mesh, each node contains a processing element (PE), and a routing element. The node consists of five input buffer queues and five output links connected by a crossbar switch. The role of the crossbar switch is to connect every input to every possible output. There is a dedicated buffer queue for messages that arrive from the PE, and two dedicated buffers per dimension, so in this case two buffers for the x dimension and two for the y dimension (one per direction). The outputs depict the direction in which the messages can travel, the messages can travel either forward, or in reverse in any given direction. When a message arrives at its destination, transmission to the local PE for consumption is also considered as an output. The basic structure of a node can be seen in Figure 2.4

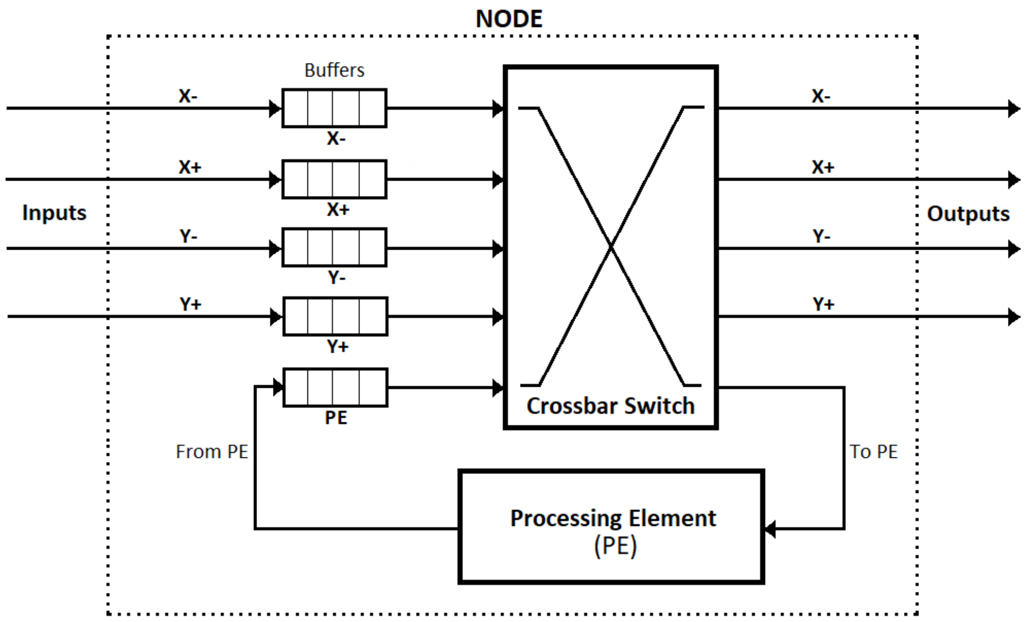


Figure 2.4 A node structure in the 2D mesh.

Switching and routing:

The nodes use packet switching with input buffer queues of large capacity. This is realistic due to Moore's Law [34], where limited memory is no longer a significant issue. In this technique messages are fully buffered at each hop. Upon arrival at a node, the message header is read and a routing decision is made to which output buffer the message is retransmitted through.

The nodes use deterministic routing to send messages to one another. Deterministic routing in the 2D mesh works as follows: a message only moves along the *x*-axis until it reaches a node with the same *x* value as the destination node. It then starts moving along the *y*-axis until it finally arrives at a node with the same *y* value as that of the destination node. For example, Figure 2.5 illustrates how a message at source (0,0) destined to the node (2,2) would first keep going along the *x*-axis until it reaches (2,0). It then goes along the *y*-axis until it reaches the destination node (2,2).

The main advantage of using deterministic routing is its ease of implementation compared to adaptive routing [19], and more importantly it avoids the issue of deadlock during message routing.

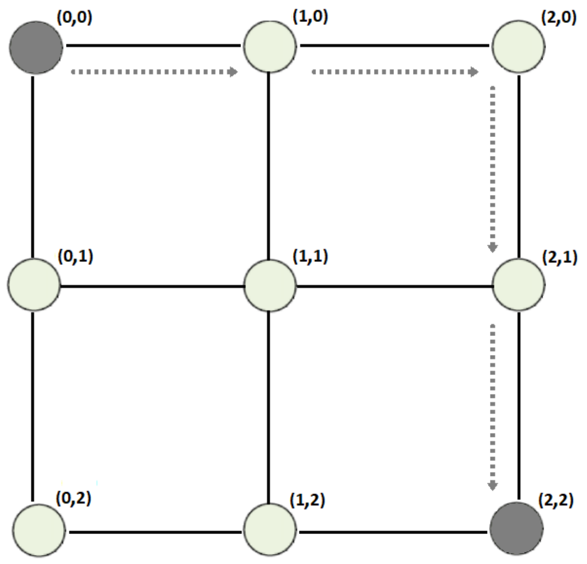


Figure 2.5 An example of a message route from source to destination using deterministic routing in the 2D mesh.

2.2.4 Conclusions

We have in this chapter presented the argument for choosing simulation as a tool to conduct our study. After that we have reviewed the different types of simulations including Monte-Carlo, trace-driven and discrete-event simulation. We then outlined the system model, including the node model and the switching and routing techniques used in our study.

The subsequent chapter presents the implementation of the simulation model for the 2D mesh using discrete-event simulation. The model is then extended to include the other topologies that are used in the comparison notably the unidirectional and bidirectional torus.

Chapter 3: Implementation of the simulation model

This chapter provides an overview of the implementation of the simulation model. The coding of the program is written using C in the Codeblocks IDE. In this section we present pseudo code for the main program of the simulator along with a description of the events involved in the simulation. The events and pseudo code described below applies to the 2D mesh, however the other topologies and their higher dimensional variants all share similar characteristics except with differences in some events, and the higher dimensional topologies also have larger data structures. The nodes are each given a distinct address (*x*,y) where x<N and y<M. N and M being the number of nodes on the x and y dimensions respectively. Our description is kept at an abstract level as much as possible for the sake of clarity for the reader; Much coding details such as the linked lists, the manipulation of the associated pointers and the models of queues have not been included due to space limitations.

3.1 Data structures

The simulation model uses different structures (which are mostly built on dynamic linked lists).

Event: This data structure stores information related to a single event in this simulation. It contains a field for the type of the event, its time, the location (i.e. the node), the input and the output.

***EventQueue*:** This is a queue of type Event that stores events that occur in this simulation. The events in the queue are sorted in order of time, with events sooner to occur given higher priority for processing.

***Message*:** This data structure stores information related to a single message. It contains a field for the ID of the message, a field for its time of arrival and a field for its destination.

***MessageQueue*:** This queue is similar to the EventQueue however it is used for storage of messages. Unlike the EventQueue, this queue does not sort the messages in order and operates a first-come-first-serve policy.

***Node*:** This data structure represents a single node in the 2D mesh. A node contains two MessageQueues per dimension of the topology and an additional queue for the PE, thus totaling 5 queues in the case of the 2D mesh. The entire topology is represented as matrix of size NxM nodes. The node also contains 5 arrays, one associated with each output. These outputs are used to store requests for these outputs when the outputs are busy.

3.2 Simulation events

The simulator program is composed of the following events:

- **Arrival:** Primary event to generate the traffic load on the network.
- **DecideRoute:** Conditional event that is used to route messages to their destination.
- **StartTransmit:** Conditional event that occurs at the start of the transmission of a message over a given output link.
- **EndTransmit:** Conditional event that occurs at the end of the transmission of a message.

Each event is associated with a procedure that describes how the event changes the state variables and advances time, possibly generating other events. In addition to these procedures, there is a procedure for initialisation that is called once at the start of the program for initialisation of the system variables (including the status of the output links, the buffers, the queues etc.). There is also a procedure for collection of statistics such as the mean response time and throughput. In what follows we will describe the procedures mentioned in more detail.

3.2.1 Initialisation

```
Procedure Initialisation() {
    Tnow = 0;
    Set lambda;
    Initialize EventQueue;
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++) {
            Initialize PE queue;
            Initialize x queues;
            Initialize y queues;
            Set request for PE output to idle;
            Set request for x output to idle;
            Set request for Xprev output to idle;
            Set request for y output to idle;
            Set request for Yprev output to idle;
        }
    }
    Schedule Arrival at t=Tnow
}
```

The system variables are initialized. Tnow is the global clock which is set to 0 (i.e. the start of the simulation time).

3.2.2 Main

```
Initialisation ();
while (Total number of transferred messages < max) do
    Get event from the event queue;
    Tnow = time of event;

    case type of event of:
        Arrival: Arrival (x,y);
        DecideRoute: DecideRoute (x,y,input);
        StartTransmit: StartTransmit (x,y,input,output);
        EndTransmit: EndTransmit (x,y,input,output);
    end case;
end while;
ReportStatistics ();
End;
```

In the main program, a call is made to the initialisation procedure to initialize the state variables. After that the program fetches events from the event queue, updates the global simulation time *tnow* and then calls the procedure associated with the event. This is repeated until a certain number of messages have reached their destination.

3.2.3 Arrival

```
Procedure Arrival(x,y) {
    Create message;
    Select random destination for message;
    Place message in PE queue;
    if (message at the head of the queue)
```

```
        Schedule DecideRoute at t=Tnow
    end if;
    Schedule Arrival at t = Tnow - (lambda*log(1-r));
}
```

This procedure generates a message in node (x,y). The destination of the message is randomly selected according to the traffic pattern used. The message is then placed in the PE queue of node (x,y). If the message is at the head of the PE queue a DecideRoute is scheduled at time=tnow. The scheduling of the following arrival in that node then follows which occurs at time=tnow-(lambda*ln(1-r)) where lambda represents the mean arrival time and r is a random number uniformly generated between 0 and 1.

3.2.4 DecideRoute

```
Procedure DecideRoute (x,y,input) {
    Check the head of InputQueue;
    if (DestinationX of message at the head of InputQueue = x and DestinationY of message
    at the head of InputQueue = y) {
        if (StatusPE=idle) Set a request for PE output;
        else
        Schedule event StartTransmit with t = Tnow with x=x,y=y, input=input and
        output=x;
    }
    else if (DestinationX of message > x) {
        if (StatusX=idle) Schedule event StartTransmit with t= Tnow with x=x, y=y,
        input=input and output=x;
        else
        Set a request for x output;
    }
    else if (DestinationX of message < x) {
        if (StatusXprev=idle) Schedule event StartTransmit with t= Tnow with x=x, y=y,
        input=input and output=Xprev;
        else
```

```
                Set a request for Xprev output;
        }
        If (DestinationY of message > y) {
                if (StatusY=idle) Schedule event StartTransmit with t=Tnow with x=x y=y
                input=input and output=y;
                else
                Set a request for y output;
        }
        If (DestinationY of message < y) {
                if (StatusYprev=idle) Schedule event StartTransmit with t=Tnow with x=x y=y
                input=input and output=Yprev
                else
                Set a request for Yprev output;
        }
}
```

Once the message is at the head of a queue and is ready for transmission, DecideRoute selects an appropriate output for the message by comparing the address of the current node to that of the destination node. Once the appropriate output is determined the procedure checks for whether the output link is idle or not. In the case of the channel being idle a StartTransmit is scheduled with time=Tnow with the output set to the chosen output. If the output is busy the message registers a request to that output.

3.2.5 StartTransmit

```
Procedure StartTransmit (x,y,input,output) {
        if (output=PE) {
                StatusPE =busy;
        }
        if (output=x) {
                StatusX=busy;
        }
```

```
    if (output=y) {
        StatusY =busy;
    }
    if (output=Xprev) {
        StatusXprev=busy;
    }
    if (output=Yprev) {
        StatusYprev=busy;
    }
}
```

After the output is chosen this procedure prepares the message for transmission by setting the appropriate output to busy. EndTransmit is then scheduled with time= Tnow + transmission time of the message.

3.2.6 EndTransmit

```
Procedure EndTransmit(x,y,input,output) {
    if (output=PE) {
        if (input=x) {
            Remove the message from x queue of node(x,y);
            Collect response time of message;
            Free the message;
        }
        if (input=Xprev) {
            Remove the message from Xprev queue of node(x,y);
            Collect response time of message;
            Free the message;
        }
        if (input=y) {
            Remove the message from y queue of node(x,y);
```

```
                Collect response time of message;
                Free the message;
            }

            if (input=Yprev) {
                Remove the message from Yprev queue of node(x,y);
                Collect response time of message;
                Free the message;
            }
            Check requests for PE for node(x,y);
            if (there is request for output PE from node(x,y) from x or Xprev or y or Yprev) {
                Schedule StartTransmit with input from the requested input queue and
                with output PE;
            }
        }
        if (output=x) {
            if (input=PE) {
                Remove the message from PE queue of node(x,y);
                Place message in x queue of node(x+1,y);
                if (message is at the head of Xqueue in node(x+1,y) {
                    Schedule DecideRoute with input from the requested input queue
                    and with output x;
                }
            }
            if (input=x) {
                Remove the message from x queue of node(x,y);
                Place message in x queue of node(x+1,y);
                if (message is at the head of Xqueue in node(x+1,y) {
```

```
                Schedule DecideRoute with input from the requested input queue
                and with output x;
            }
        }
        if (there is request for output x from node(x,y) from PE or x){
            Schedule StartTransmit with input from the requested input queue and
            with output x;
        }
    }

    if (output=Xprev) {
        if (input=PE) {
            Remove the message from PE queue of node(x,y);
            Place message in Xprev queue of node(x-1,y);
            if (message is at the head of Xprevqueue in node(x-1,y) {
                Schedule DecideRoute with input from the requested input queue
                and with output Xprev;
            }
        }
        if (input=Xprev) {
            Remove the message from Xprev queue of node(x,y);
            Place message in Xprev queue of node(x+1,y);
            if (message is at the head of Xprevqueue in node(x+1,y) {
                Schedule DecideRoute with input from the requested input queue
                and with output Xprev;
            }
        }
        if (there is request for output Xprev from node(x,y) from PE or x) {
            Schedule StartTransmit with input from the requested input queue and
            with output Xprev;
        }
    }
    if (output=y) {
        if (input=PE) {
            Remove the message from PE queue of node(x,y+1);
            Place message in y queue of node(x-1,y);
            if (message is at the head of Yqueue in node(x+1,y) {
                Schedule DecideRoute with input from the requested input queue
                and with output y;
            }
        }
        if (input=x) {
            Remove the message from x queue of node(x,y);
            Place message in x queue of node(x,y+1);
            if (message is at the head of Yqueue in node(x,y+1) {
                Schedule DecideRoute with input from the requested input queue
                and with output y;
            }
        }
        if (input=Xprev) {
            Remove the message from Xprev queue of node(x,y);
            Place message in y queue of node(x,y+1);
            if (message is at the head of Yqueue in node(x,y+1) {
                Schedule DecideRoute with input from the requested input queue
                and with output y;
            }
        }
        if (input=y) {
            Remove the message from y queue of node(x,y);
            Place message in y queue of node(x,y+1);
            if (message is at the head of Yqueue in node(x,y+1) {
            Schedule DecideRoute with input from the requested input queue and
            with output y;
            }
        }


        if (there is request for output y from node(x,y) from PE or x or Xprev or y) {
            Schedule StartTransmit with input from the requested input queue and
            with output y;
        }
    }
    if (output=Yprev) {
        if (input=PE) {
            Remove the message from PE queue of node(x,y);
            Place message in Yprev queue of node(x,y-1);
            if (message is at the head of Yprevqueue in node(x,y-1) {
                Schedule DecideRoute with input from the requested InputQueue
                and with output Yprev;
            }
        }
        if (input=x) {
            Remove the message from x queue of node(x,y);
            Place message in Yprev queue of node(x,y-1);
            if (message is at the head of Yprevqueue in node(x,y-1) {
                Schedule DecideRoute with input from the requested InputQueue
            and with output Yprev;
            }
        }
        if (input=Xprev) {
            Remove the message from Xprev queue of node(x,y);
            Place message in Yprev queue of node(x,y-1);
            if (message is at the head of Yprevqueue in node(x,y-1) {
                Schedule DecideRoute with input from the requested input queue
                and with output Yprev;
            }
        }
        if (input=Yprev) {
            Remove the message from Yprev queue of node(x,y);
            Place message in Yprev queue of node(x,y-1);
            if (message is at the head of Yprevqueue in node(x,y-1) {
                Schedule DecideRoute with input from the requested input queue
                and with output Yprev;
            }
        }

        if (there is request for output y from node(x,y) from PE or x or Xprev or Yprev) {
            Schedule StartTransmit with input from the requested input queue and
            with output Yprev;
        }
    }
}
```

This procedure moves the message from the input queue of the sender node and puts it in the input queue of the next node along the path. If that message is at the head of the queue, the event DecideRoute is scheduled at time=tnow. If the message is at the final destination, the message is sent to the local PE where statistics are collected and the message record destroyed. The output link then checks for requests from any messages that are waiting to use the output link. If any requests are found, a StartTransmit is scheduled at time=tnow.

3.3 Model validation

To validate the simulation model, the simulation was run over a number of smaller, easily predictable cases. For instance, in the 2D mesh topology under low traffic the response time for a message can be given by [3]

$$D \times L$$

Where D is the number of hops between the sender and destination nodes and L is the message length. For instance, a message of length 32 phits sent from node (0,0) to (2,2) would take 4 hops and as a consequence the response time for the message would be 4x32=128 cycles. We compare this value against that supplied by the simulation to check agreement. We have found that in all tested cases under light traffic and moderate traffic, the agreement between the calculated result and the results from simulation are in satisfactory agreement.

3.4 Conclusions

In this chapter, the implementation of the simulation model, including the simulation events and data structures, for the 2D mesh topology has been described. The program for the simulation model has been described in pseudocode. The different events including Arrival, DecideRoute, StartTransmit and EndTransmit describe the operation of the network starting from the time a message is generated in a node with a given destination and placed in a local buffer for transmission from node to node until it arrives at its destination. The simulation model has been specified for deterministic routing and packet switching.

The following chapter presents the various parameters and assumptions used in this work, then introduces the various different comparisons carried out along with the presentations of results and discussion.

Chapter 4: Performance comparison of network-on-chip topologies

In this chapter, we will use the simulation model described in Chapter 4 to carry out the performance comparison between the well-known NoC topologies namely the mesh, the torus with its unidirectional and bidirectional variants. In the first stage, the comparison is carried out assuming no technological constraints imposed on system implementation. The three topologies are compared in their 2D and 3D versions. Both the uniform traffic and hotspot traffic patterns have been considered in the comparison. In the second stage, the same performance comparison has been conducted between these topologies taking into account the physical constraints imposed by the implementation technology, notably the bisection width which is relevant to the implementation of NoCs in VLSI technology [12, 13].

In what follows, we will start by outlining the assumptions used in this study, then describing the method for collecting the simulation results. After that we present the performance results along with discussions.

4.1 Assumptions

The assumptions which have been used throughout this simulation study have widely adopted in existing studies [24, 25]:

1) Message generation at a node is independent of all other nodes.
2) The message arrival rate at each node follows a Poisson distribution with a mean inter-arrival rate 1/λ messages/cycle. Thus, the message inter-arrival time follows an exponential distribution with a mean arrival time λ cycles.
3) The generated messages are of fixed length.
4) Propagation delay across the links is negligible.
5) Routing time (time for a router to decide which output to select for a given message) is negligible.
6) No nodes or links break down during the simulation.

4.2 Simulation parameters

The simulation model includes various parameters described below.

4.2.1 Traffic pattern

The traffic pattern in general describes how nodes communicate among each other. In this study we have used two traffic patterns, namely uniform and hotspot. These traffic patterns have been widely used in existing comparison studies [23, 24, 25, 27, 28].

- **Uniform:** Each node has an equal probability of sending a message to any other node.
- **Hotspot:** Nodes favor sending messages to a specific node with probability p, other messages are sent following a uniform traffic pattern.

4.2.2 Message arrival rate

This parameter refers to the number of messages that can be produced during a given period of time. The arrival rate is gradually changed to reflect the network operating different operating conditions including light, moderate and heavy traffic.

4.2.3 Network size

The network size is a parameter which is useful for evaluating the properties of networks such as scalability. In this study, we examine different network sizes including 8x8 and 32x32 nodes for the 2D topologies, and 4x4x4 and 10x10x10 nodes for the 3D topologies. Due to limitations in computing resources and time, larger network sizes could not be examined.

4.3 Performance metrics

The comparison among the different topologies have been based on the following performance metrics

4.3.1 Mean response time

The response time is a qualitative measure of network performance. The response time for a single message is the elapsed time from sending a message from a source node until it arrives at its destination node. The response time is measured in number of cycles, where a cycle is the amount of time to send a phit across a link.

$$R = t_{now} - t_{arrival}$$

Where t_{now} is the time at which a message reaches the destination node, and $t_{arrival}$ is the time of generation of the message at the source node. The mean response time is then found by averaging the response time over all delivered messages. This can be written as

$$\bar{R} = \frac{\sum_{i=1}^{N} R_i}{N}$$

Where R_i is the response time of an individual message *i*.

4.3.2 Mean throughput

Throughput is the average amount of messages delivered per unit of time. This is a quantitative measure of network performance that describes the raw output of the network. In this case it is measured in the number of messages per cycle. This is given by

$$Th = \frac{Total\ messages\ delivered}{simulation\ time}$$

4.4 Batch means method for result collection

In the batch means method [29], the simulation is run once over an extended period of time. The simulation is then divided into a number batches of a specific number of delivered messages (let's say 10000), and measures are collected over each batch to form a single point estimate of the performance measure of interest (e.g. response time or throughput).

As every message arrives at its destination, the response time for that message is collected and added to the overall response time, and a counter that tallies the total number of messages delivered is incremented. When a batch is completed, the mean response time and throughput for that specific batch are calculated and stored. The variables related to the collection of statistics are then reset without stopping the running of the simulation. The same process is repeated for each batch, and after the last batch is complete the overall mean (mean of the means of the batches) for both the response time and throughput are computed. The batch means method is useful because it enables us to ensure that the results reflect the system in a steady-state behavior. Furthermore, it allows us to avoid the warm-up effect on the simulation results [29].

4.5 Confidence interval

The confidence interval calculates the range for the possible values of a specific performance measure. That is, for a 95% confidence for instance, if the simulation were to be run 100 times, the mean of that performance measure would fall within that interval on 95 occasions [29]. In order to calculate the confidence interval for a given performance measure, the overall mean and the standard deviation must be computed. The overall mean is given by

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

where x_i is the mean of a single batch and *N* is the total number of batches. On the other hand, the standard deviation calculates the average distance of each batch mean from the overall mean, and it is given by [29]

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}}$$

Once the standard deviation is found the confidence interval is then found by

$$\bar{x} \pm 1.96\,\frac{s}{\sqrt{n}}$$

4.6 Results and Discussion

This section presents first the simulation results for comparing the three topologies under unconstrained and constrained implementations, followed by the results of the 2D vs 3D comparison under constrained and unconstrained implementations.

4.6.1 Mesh vs unidirectional torus vs bidirectional torus: Unconstrained implementation

In this case, the topologies are not be subjected to any physical constraints imposed by implementation technology. As a consequence, we assume that the different network topologies all have the same channel width (i.e. channel bandwidth) irrespective of the network size. This enables us to assess the impact of the graph-theoretical properties of the various topologies on system performance.

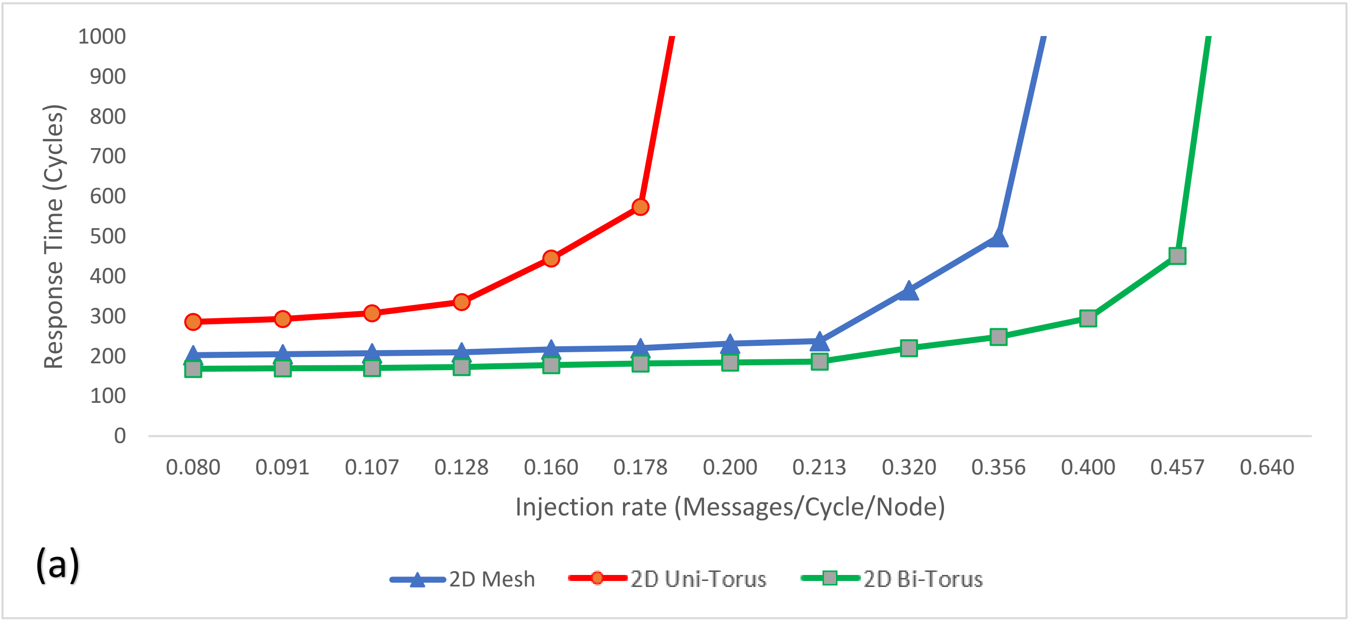
Scenario 1: Uniform traffic

In this scenario, the mesh and the unidirectional as well as the bidirectional torus are subjected to various traffic rates using the uniform traffic pattern, where a sender node has an equal probability of sending a message to any other destination node.

Before presenting the performance results for the three topologies, we show first in Table 4.1 for the sake of illustration the confidence intervals obtained for the response time and throughput for the 2D mesh with 8x8 nodes. Each performance result in the table and in all the figures below has been collected from 10 batches where each batch reflects the statistics of at least 2500 delivered messages. However, the results for the confidence interval will not be shown for the other scenarios and topologies for the sake of clarity of the figures and due to space limitations.

Table 4.1 Results for the confidence interval for the 8x8 2D mesh.

Injection rate	Lower bound response time	Upper bound response time	Lower bound throughput	Upper bound throughput
400	202.4	203.6	0.156	0.16
350	203.1	204.4	0.172	0.178
300	206	207.3	0.207	0.211
250	208.8	210	0.245	0.251
200	217	219	0.312	0.319
180	219.5	221.7	0.338	0.346
160	227.5	230	0.385	0.395
150	233.8	236.8	0.414	0.422
100	349.1	367.2	0.623	0.637
90	481.2	507.8	0.674	0.688
80	2178.2	2981.8	0.738	0.753



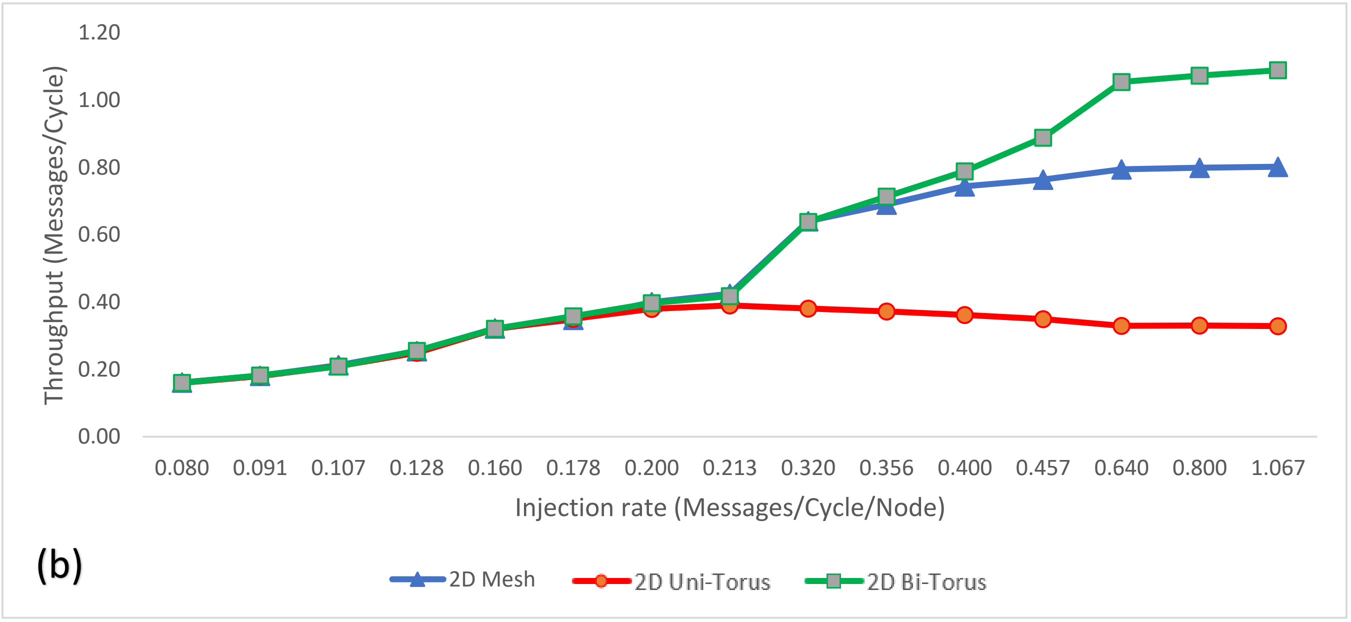
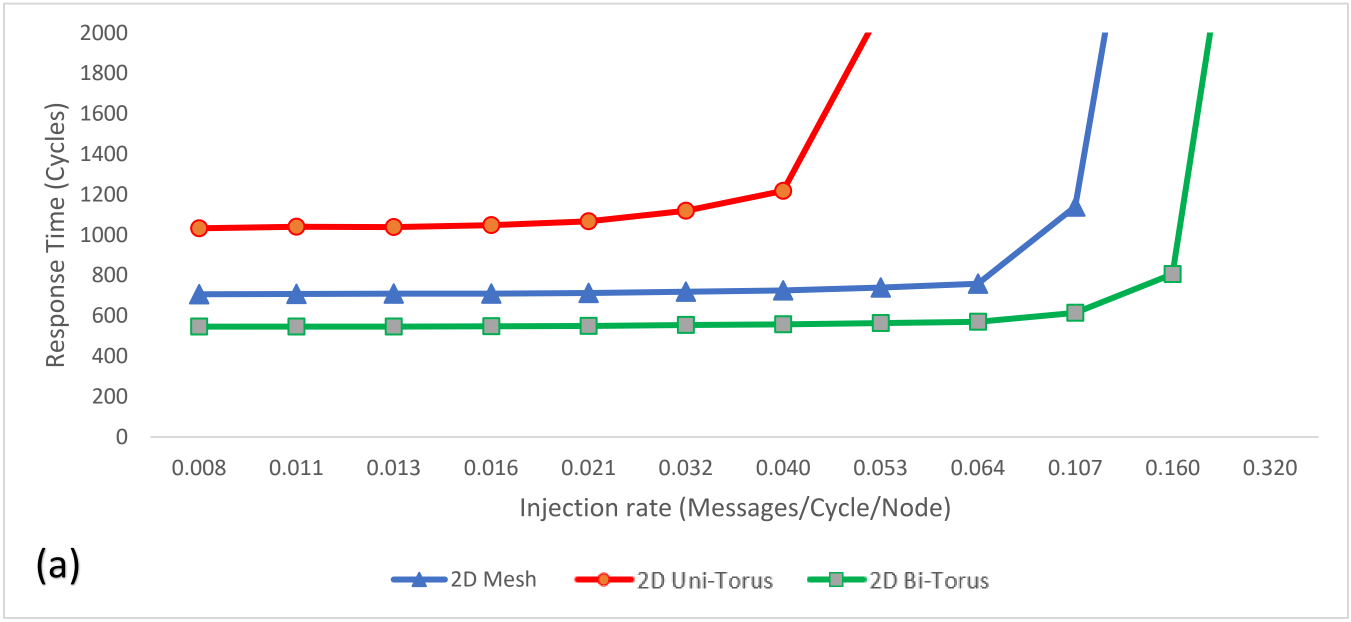


Figure 4.1 Performance results for the 2D mesh vs unidirectional vs bidirectional torus under unconstrained uniform traffic for 8x8 nodes (a) Response time, (b) Throughput.



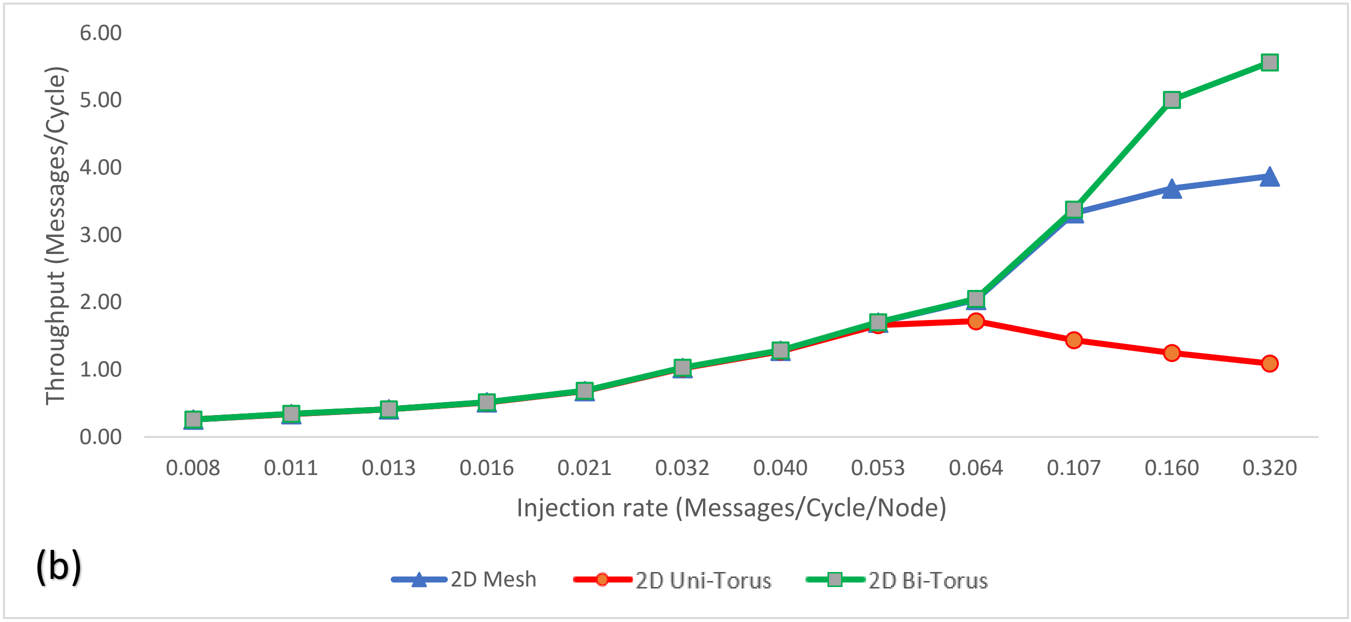
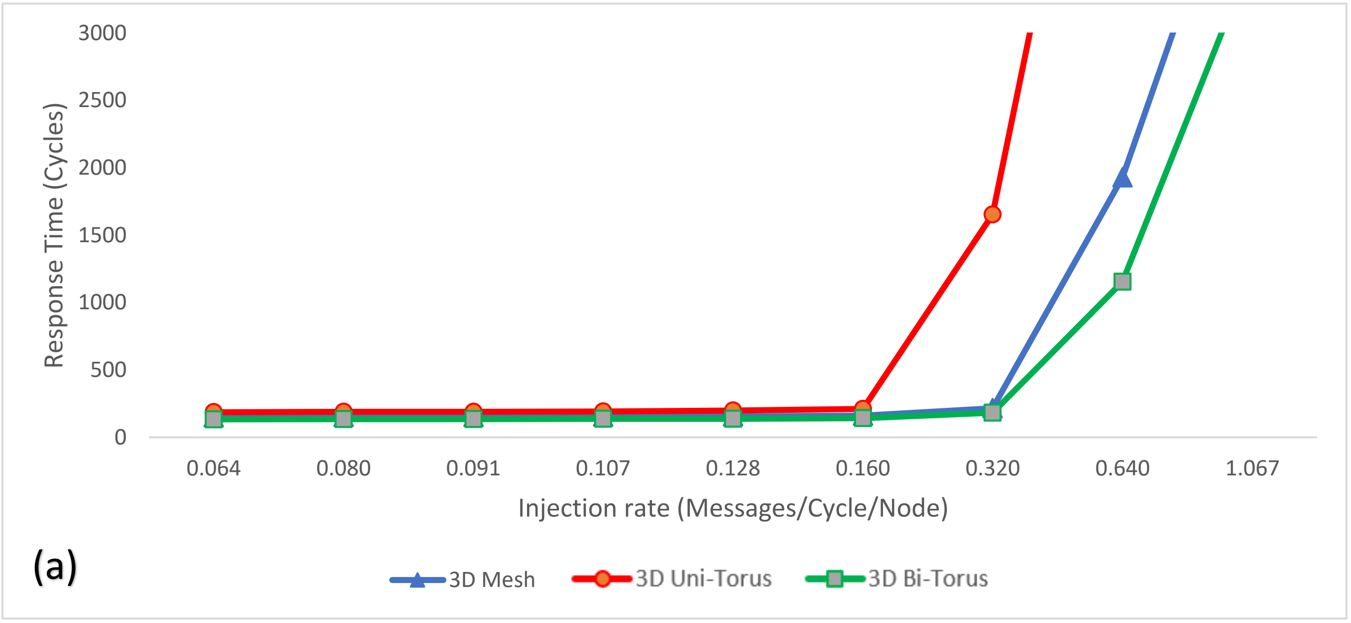


Figure 4.2 Performance results for the 2D mesh vs unidirectional vs bidirectional torus under unconstrained uniform traffic for 32x32 nodes (a) Response time, (b) Throughput.



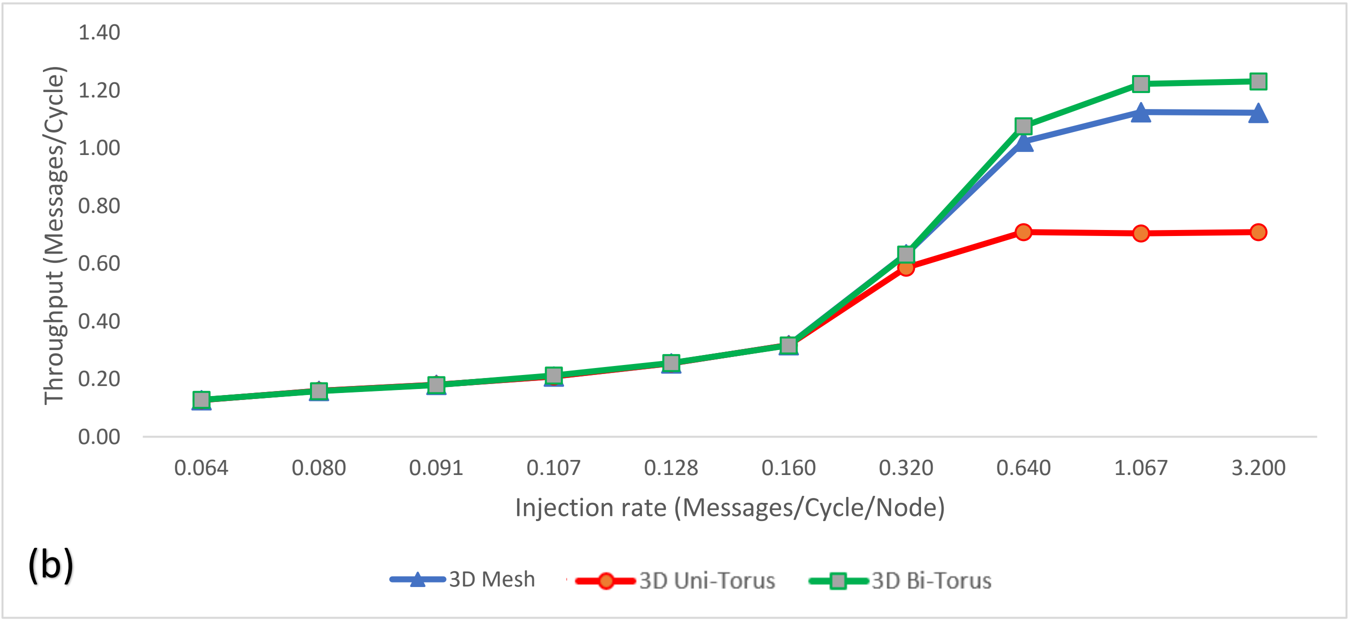
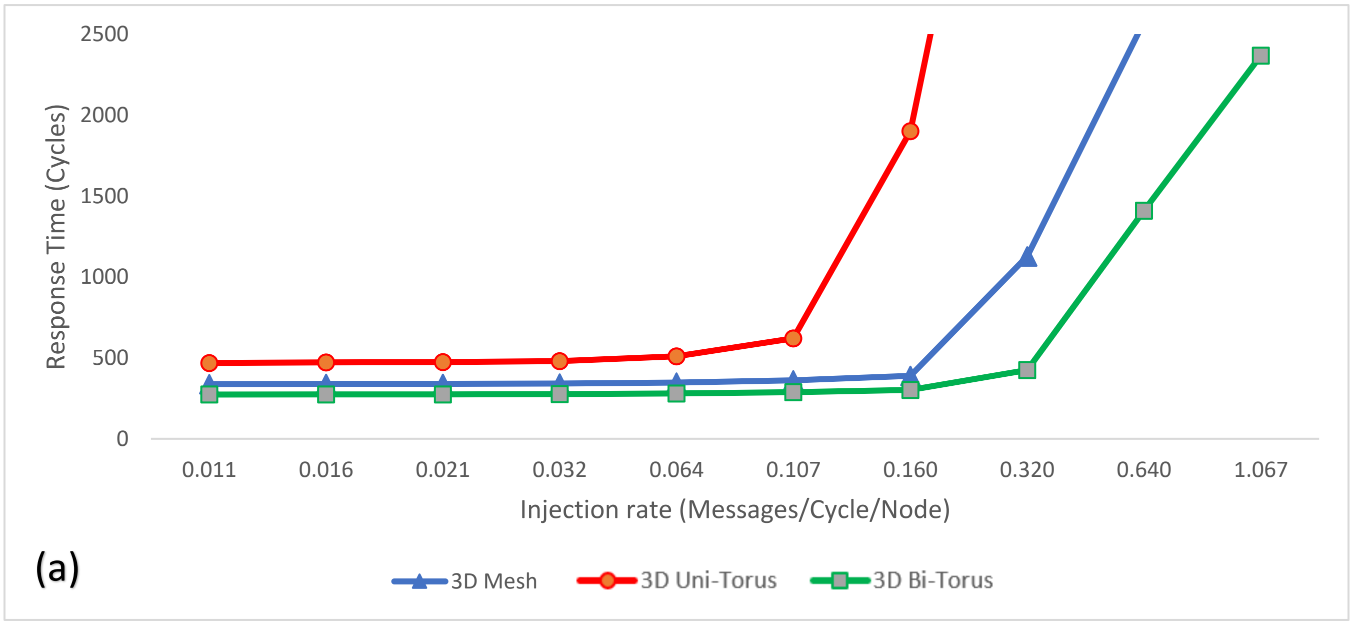


Figure 4.3 Performance results for the 3D mesh vs unidirectional vs bidirectional torus under unconstrained uniform traffic for 4x4x4 nodes (a) Response time, (b) Throughput.



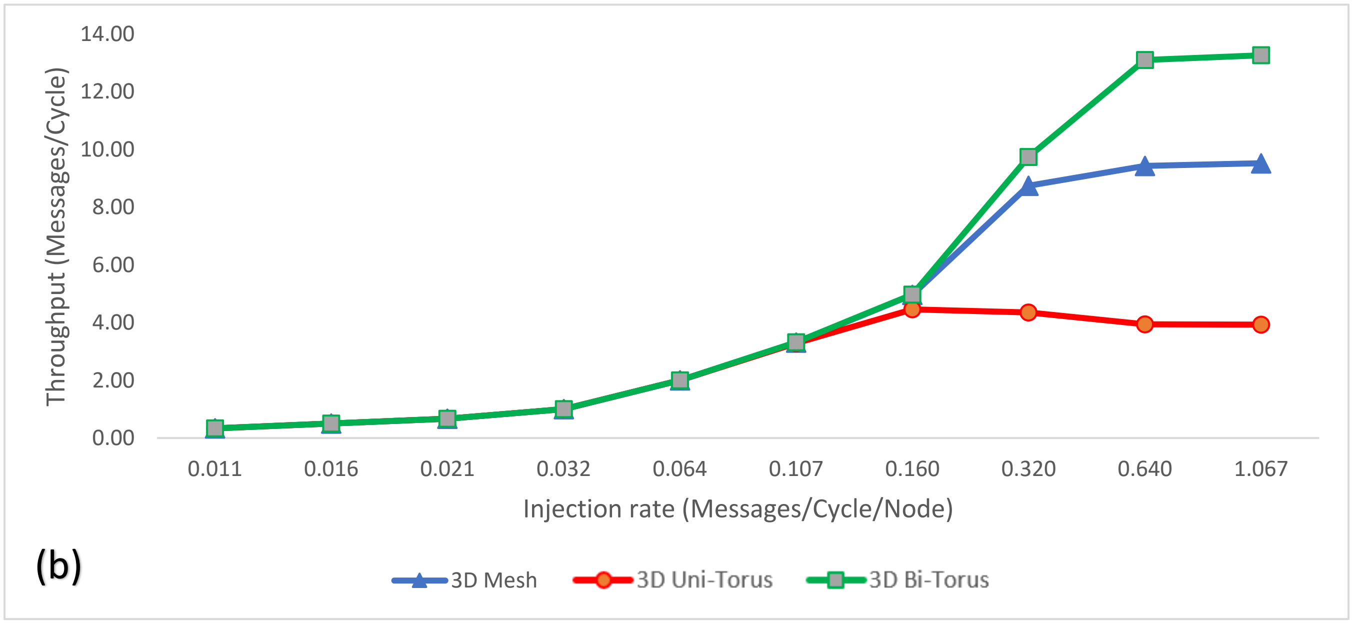


Figure 4.4 Performance results for the 3D mesh vs unidirectional vs bidirectional torus under unconstrained uniform traffic for 10x10x10 nodes (a) Response time, (b) Throughput.

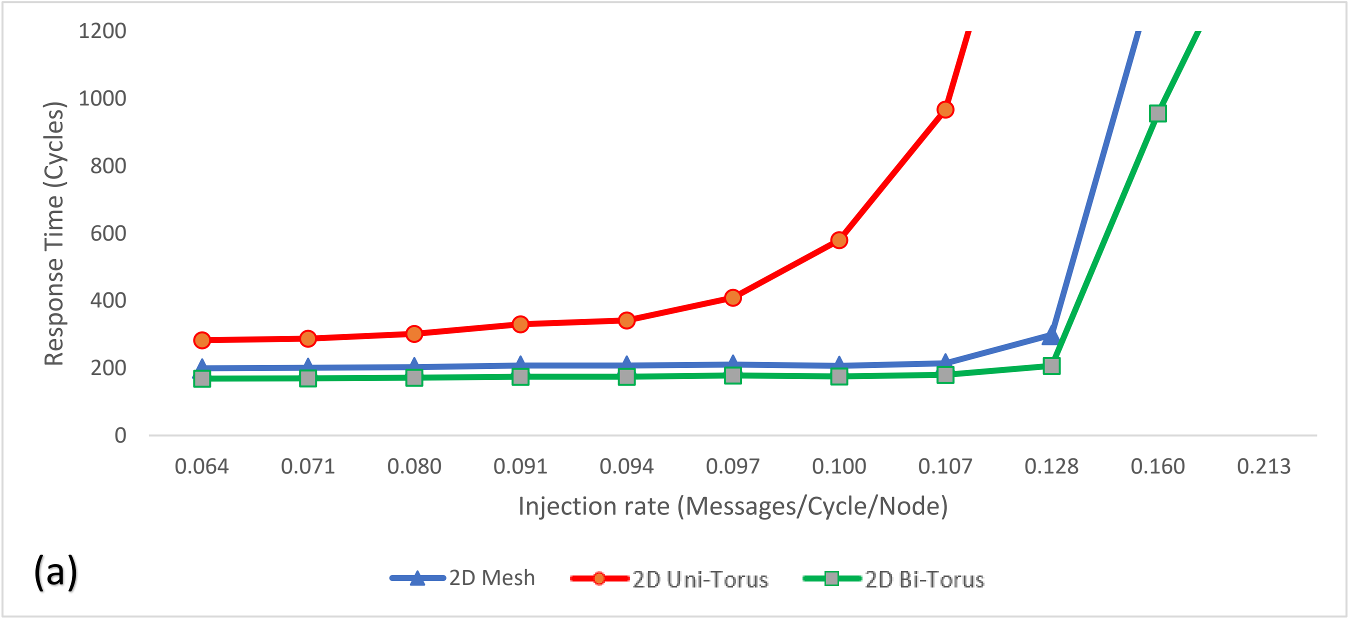
The 2D versions of the networks are compared amongst each other. The same comparison is also conducted for the 3D versions. The simulation results are depicted below for network sizes of 8x8 and 32x32 nodes for the 2D versions and 4x4x4 and 10x10x10 nodes for their 3D counterparts. In all the figures, the x-axis represents the rate of messages injected into the network (measured by messages/cycle where the cycle is the time to send a phit across a link). In the figures 4.1(a) to 4.22(a) the y-axis represents the mean message response time (measured in cycles) whereas in figures 4.1(b) to 4.22(b) the y-axis represents the mean throughput (measured in messages/cycle).

The figures 4.1 to 4.4 reveal that regardless of the network size or dimensions, similar trends in performance can be observed. The bidirectional torus exhibits the lowest response times and highest throughput under most traffic load conditions. As the bandwidth of the links is the same in all the topologies this can be attributed to the bidirectional torus' lower average message distance in comparison to the unidirectional torus. This means that messages take less time to cross from source to destination in the bidirectional torus. The mesh showing lower performance in terms of response time and throughput than the bidirectional torus can be attributed to its topological asymmetry, which results in larger amounts of traffic congestion towards the center. The disparity becomes more and more apparent with the increase in network size indicating better scalability properties of the bidirectional torus.

Scenario 2: Hotspot traffic

The same simulation experiment performed in the above Scenario 1 has been repeated considering the hotspot traffic pattern where a sender node sends a message to the hotspot node located in the center of network with probability α and a probability of 1-α to any other node with equal probability. In the figures 4.5 to 4.8 are the results for network sizes of 8x8 and 32x32 nodes for 2D and 4x4x4 and 10x10x10 nodes for the 3D networks where α is set to 0.1.

The figures reveal that the same conclusions as scenario 1 are reached in that the bidirectional torus exhibits superior performance for both response time and throughput followed by the mesh and then by the unidirectional torus. This is due to the bidirectional torus being a symmetrical topology and having a lower average message distance.



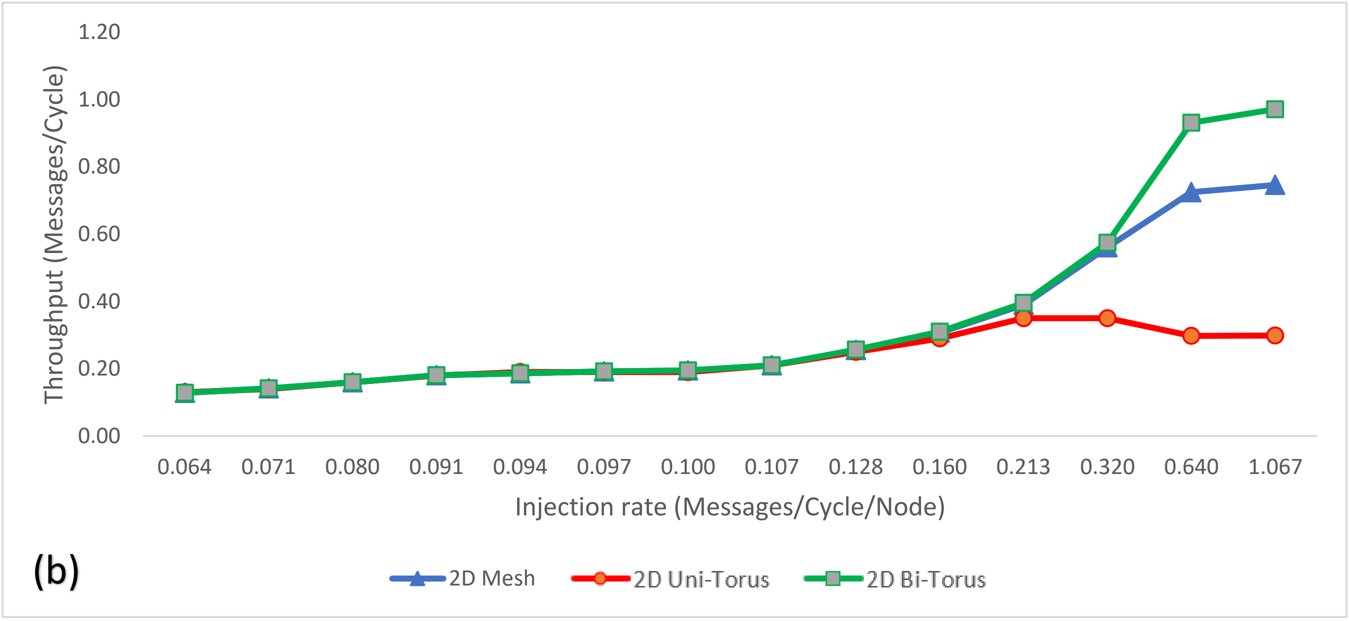
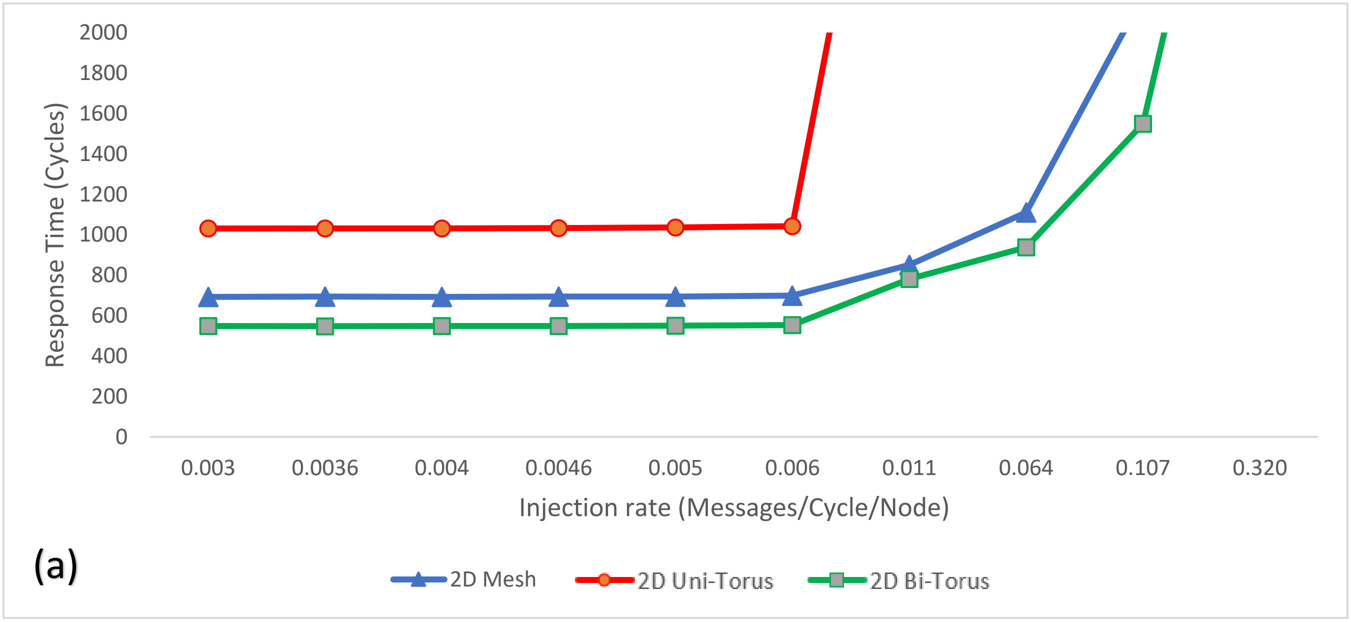


Figure 4.5 Performance results for the 2D mesh vs unidirectional vs bidirectional torus under unconstrained hotspot traffic for 8x8 nodes (a) Response time, (b) Throughput.



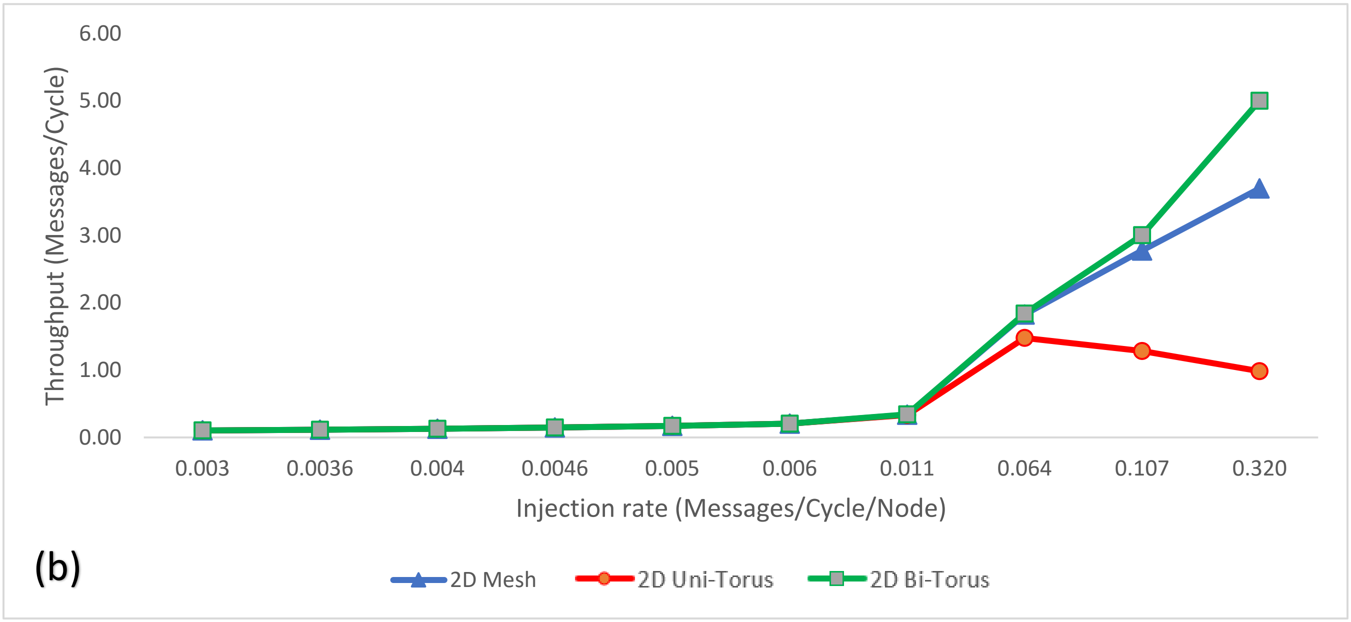
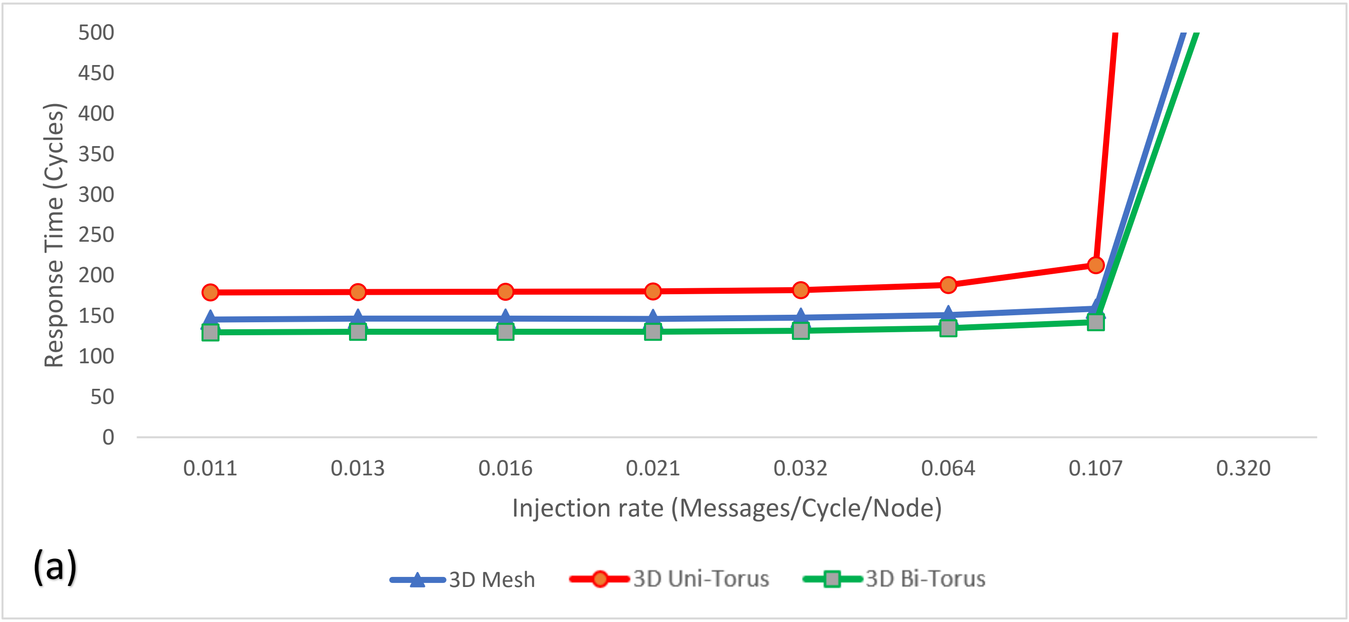


Figure 4.6 Performance results for the 2D mesh vs unidirectional vs bidirectional torus under unconstrained hotspot traffic for 32x32 nodes (a) Response time, (b) Throughput.



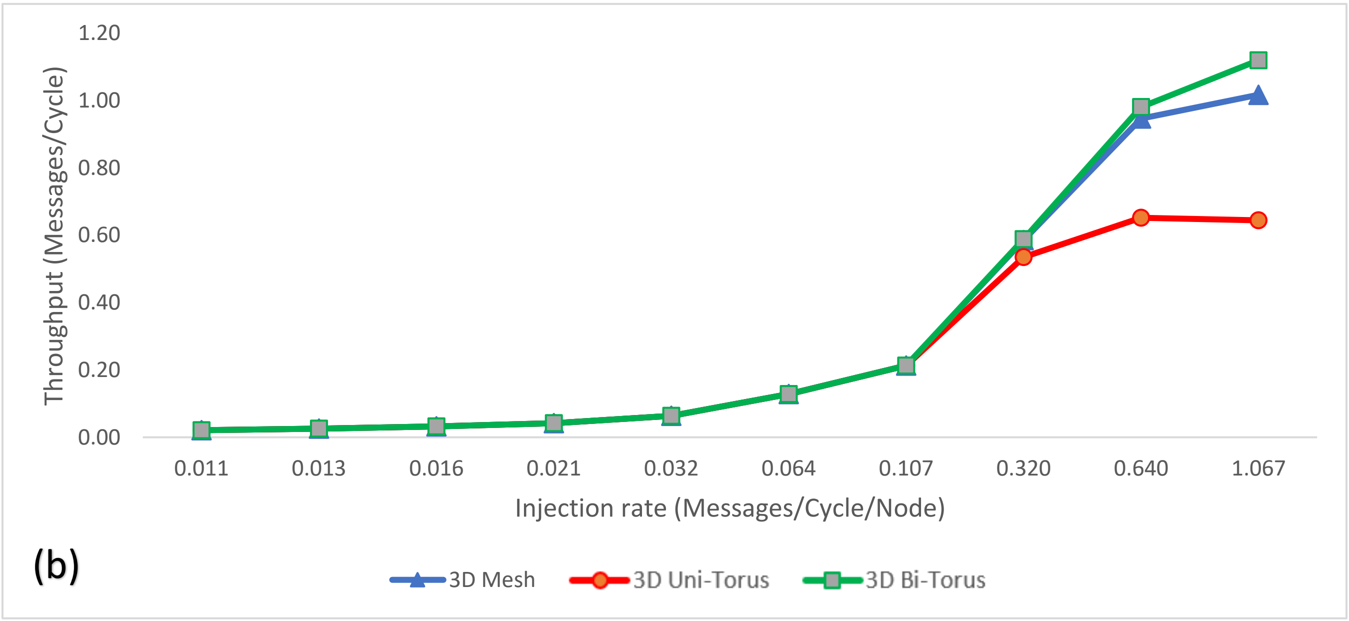
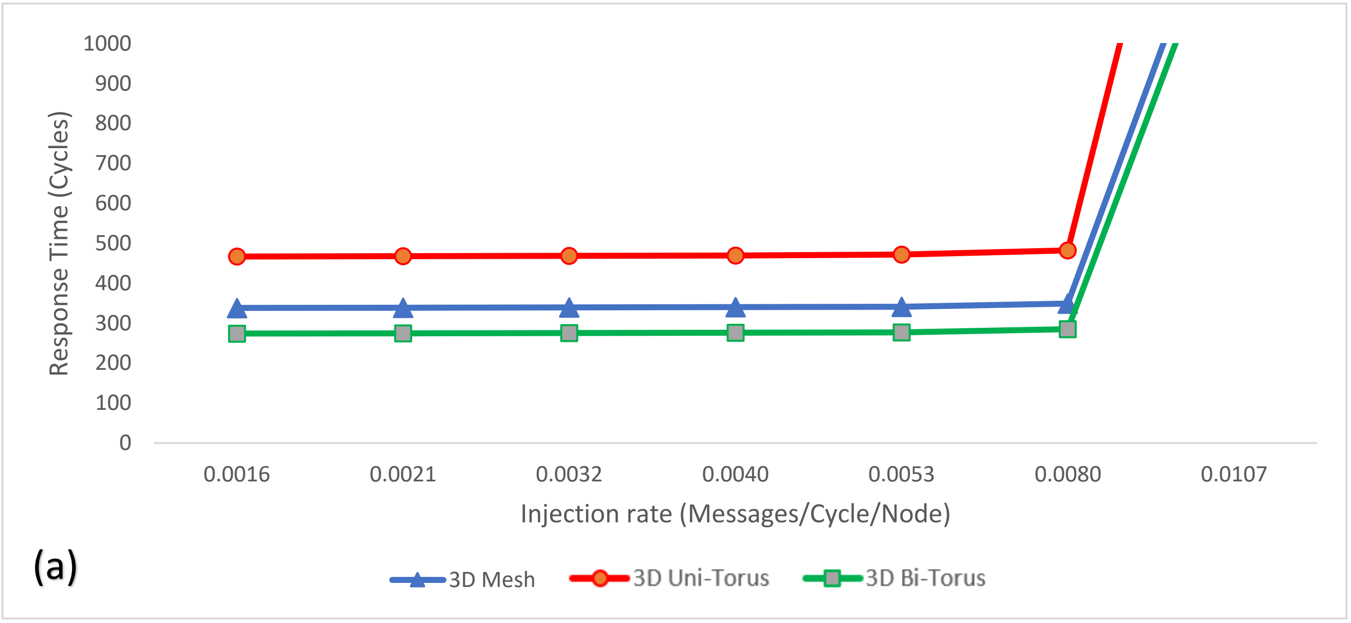


Figure 4.7 Performance results for the 3D mesh vs unidirectional vs bidirectional torus under unconstrained hotspot traffic for 4x4x4 nodes (a) Response time, (b) Throughput.



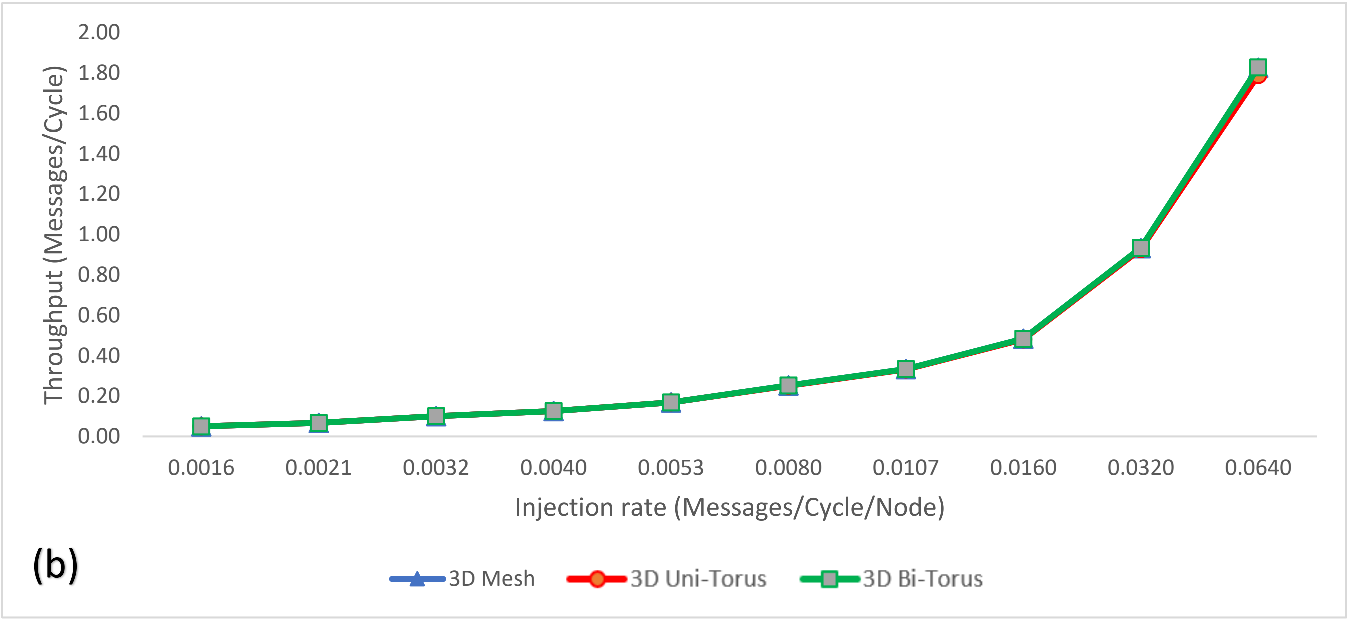


Figure 4.8 Performance results for the 3D mesh vs unidirectional vs bidirectional torus under unconstrained hotspot traffic for 10x10x10 nodes (a) Response time, (b) Throughput.

4.6.2 Mesh vs unidirectional torus vs bidirectional torus: Constrained implementation

Whilst the graph-theoretical properties of networks have great influence on system performance [35], they may not reveal the full story once systems are physically implemented onto a chip as is the case for NoCs [35]. In this study, we have used the bisection width constraint imposed by VLSI implementation technology [35, 36] to evaluate the impact of the physical implementation constraints on network performance. Incorporating such implementation constraints may lead to different conclusions to the above conclusions reached in the scenarios on "unconstrained implementation" regarding the relative performance merits of the different competing network topologies.

In order to determine the bandwidth of the links for a given network topology, we have used the bisection width that has been suggested for VLSI implementation technology [35, 36]. According to Dally's work [35], the bisection width for the mesh is given by

$$B_{mesh} = 2\sqrt{N}k^{\frac{n}{2}-1}W_{mesh}$$

with $N=k^n$ being the network size, k being the number of nodes per dimension and n being the number of dimensions and W_{mesh} being the channel width (i.e., the number of wires per link). On the other hand, Dally has found the bisection width for the bidirectional torus to be [35]

$$B_{bitorus} = \frac{4N}{k}W_{bitorus}$$

With $W_{bitorus}$ being the channel width. Given that the unidirectional torus has only one link per dimension, its bisection width is half of that of the bidirectional torus and is found to be [35]

$$B_{unitorus} = \frac{2N}{k}W_{unitorus}$$

Assuming a fixed bisection width across the three networks in order to reflect the physical constraints (In this case wiring limitations) imposed by implementation technology, it can be deduced that the channel width of a link in the bidirectional torus in terms of the channel width of that of the mesh is as follows [35]

$$B_{bitorus} = B_{mesh} \text{ , thus}$$

$$\frac{4N}{k}W_{bitorus} = 2\sqrt{N}k^{\frac{n}{2}-1}W_{mesh}$$

Therefore, the channel width of a link in the bidirectional torus $W_{bitorus}$ can be expressed in terms of that of the mesh as [35]

$$W_{bitorus} = \frac{1}{2}W_{mesh}$$

Using the same arguments, it can be shown that the channel width of a link in the unidirectional torus $W_{unitorus}$ can be written as [35]

$$W_{unitorus} = W_{mesh}$$

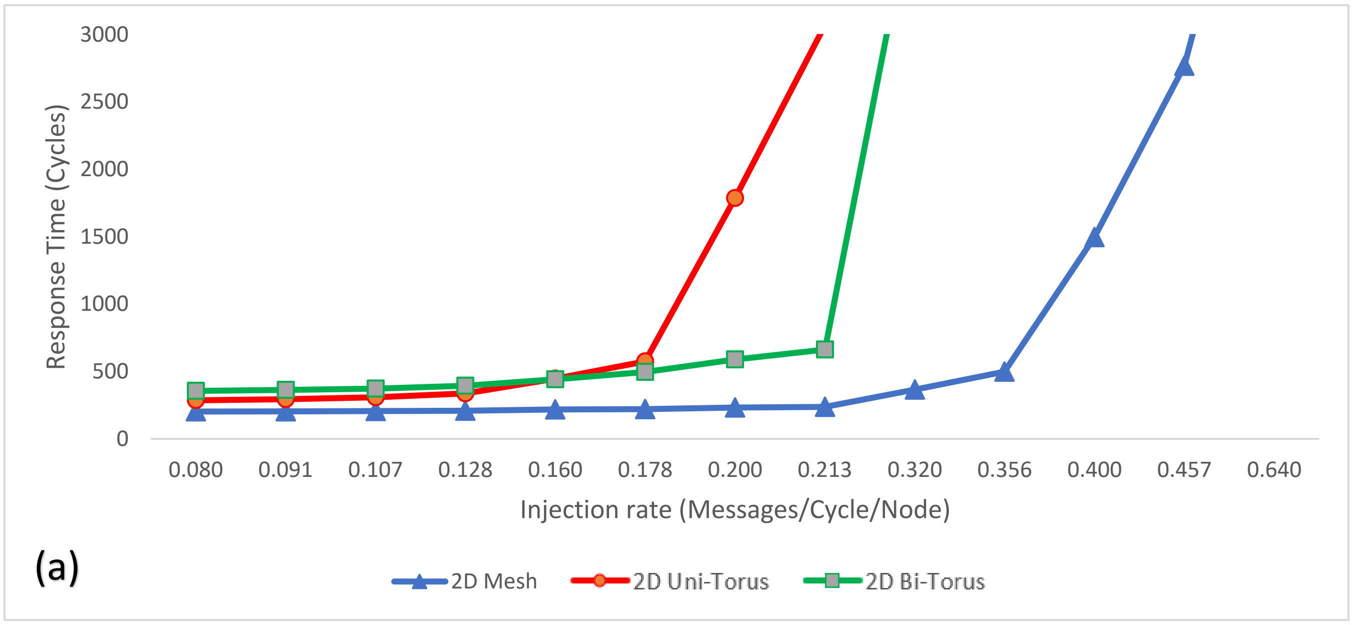
Given that the channel width (i.e. the number of wires) is directly proportional to the bandwidth (given by phits/cycle) of the link, the above equations reveal that the bandwidth of the link in the mesh and unidirectional torus are equal. However, the bandwidth of the link in the bidirectional torus is half of that of the mesh and unidirectional torus. As a result, the message length in the bidirectional torus is double the message length in the other two topologies. To illustrate this, for instance if a message is 32 phits long and therefore takes 32 cycles to be transmitted on a link in the mesh or unidirectional torus, it will take 64 cycles in the bidirectional torus. The following scenarios will compare the 2D topologies amongst each other with physical constraints taken into account over different network sizes.

Scenario 1: Uniform traffic

The three topologies are subjected to a uniform traffic pattern. The following figures depict the results of the comparisons.

In figures 4.9 to 4.10, the impact of the reduction in the channel bandwidth of the bidirectional torus can be clearly observed as it is now outperformed by the mesh across the considered network sizes in terms of both response time and throughput. This is in contrast to the performance outcome of the previous scenarios where the bidirectional torus exhibited the best performance out of the three topologies. However, the bidirectional torus still outperforms the unidirectional torus despite its reduction in channel bandwidth. This might be attributed to the difference in average message distance between the two topologies which is large enough to the point where a reduction in bandwidth is not enough to offset the difference in channel bandwidth. However, the bidirectional torus being symmetrical is not enough to compensate for its reduced channel bandwidth when compared to the mesh.

It’s worth mentioning that the performance results of the 3D versions of the three topologies have revealed similar performance trends as for the 2D case. However, the results have not been presented due to space limitations.



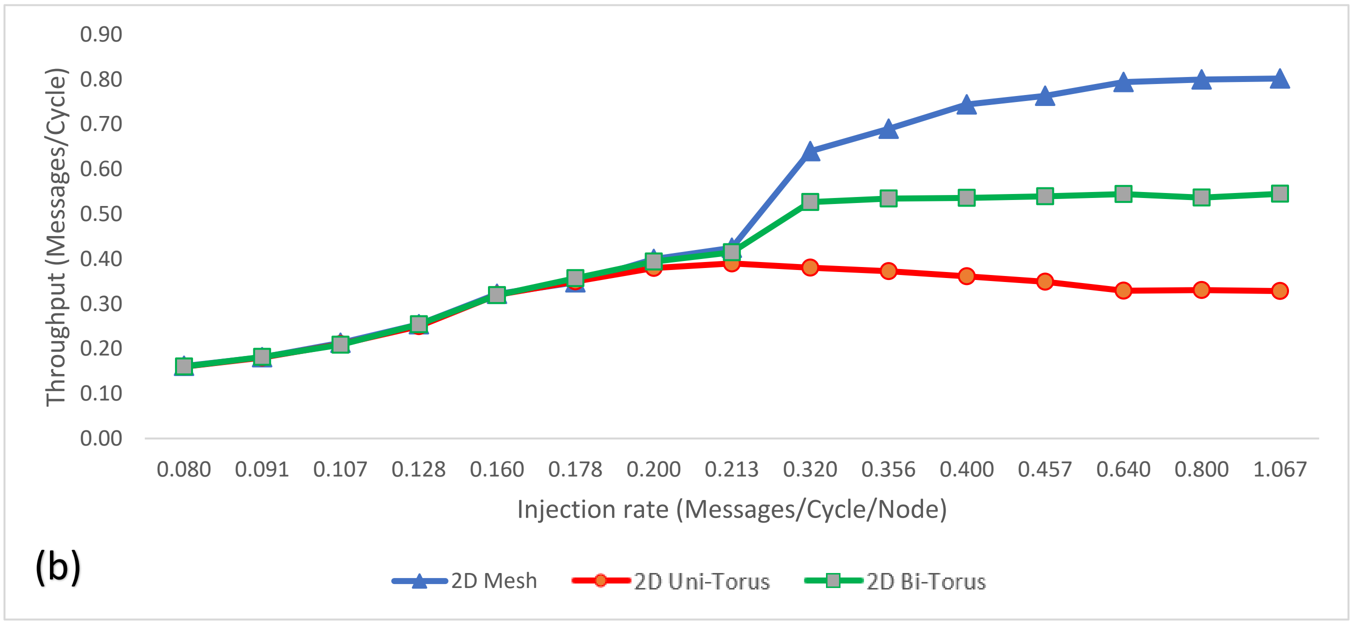
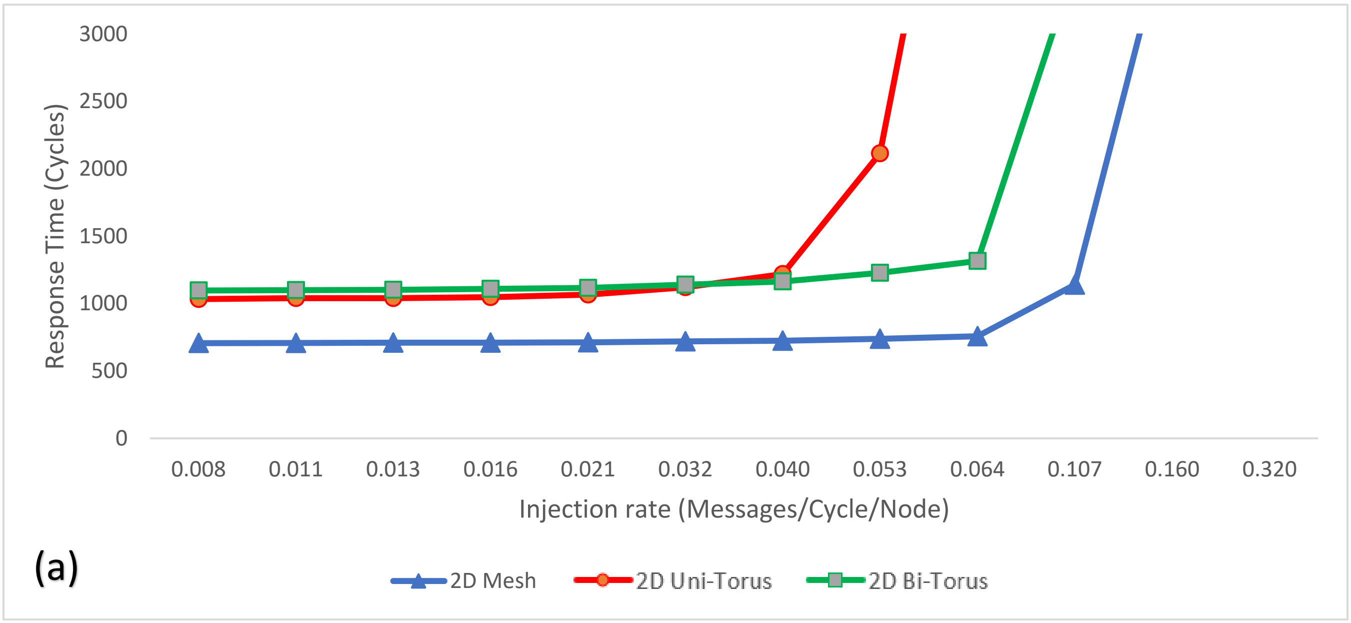


Figure 4.9 Performance results for the 2D mesh vs unidirectional vs bidirectional torus under constrained uniform traffic for 8x8 nodes (a) Response time, (b) Throughput.



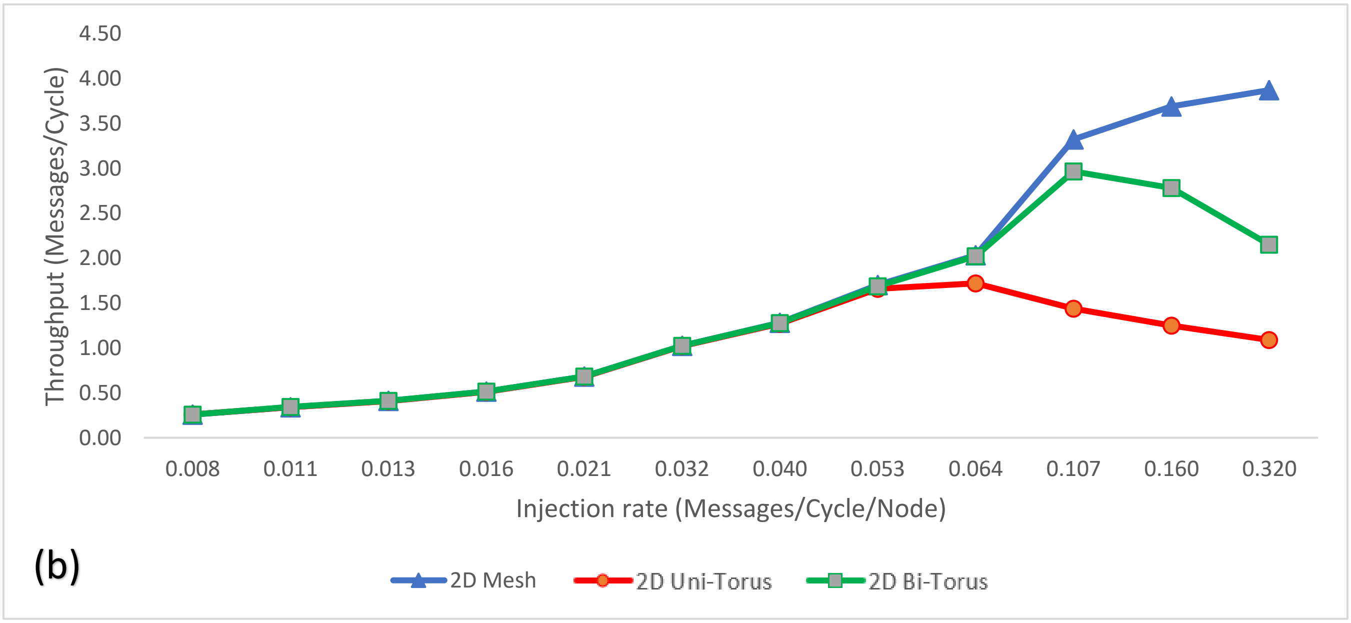


Figure 4.10 Performance results for the 2D mesh vs unidirectional vs bidirectional torus under constrained uniform traffic for 32x32 nodes (a) Response time, (b) Throughput.

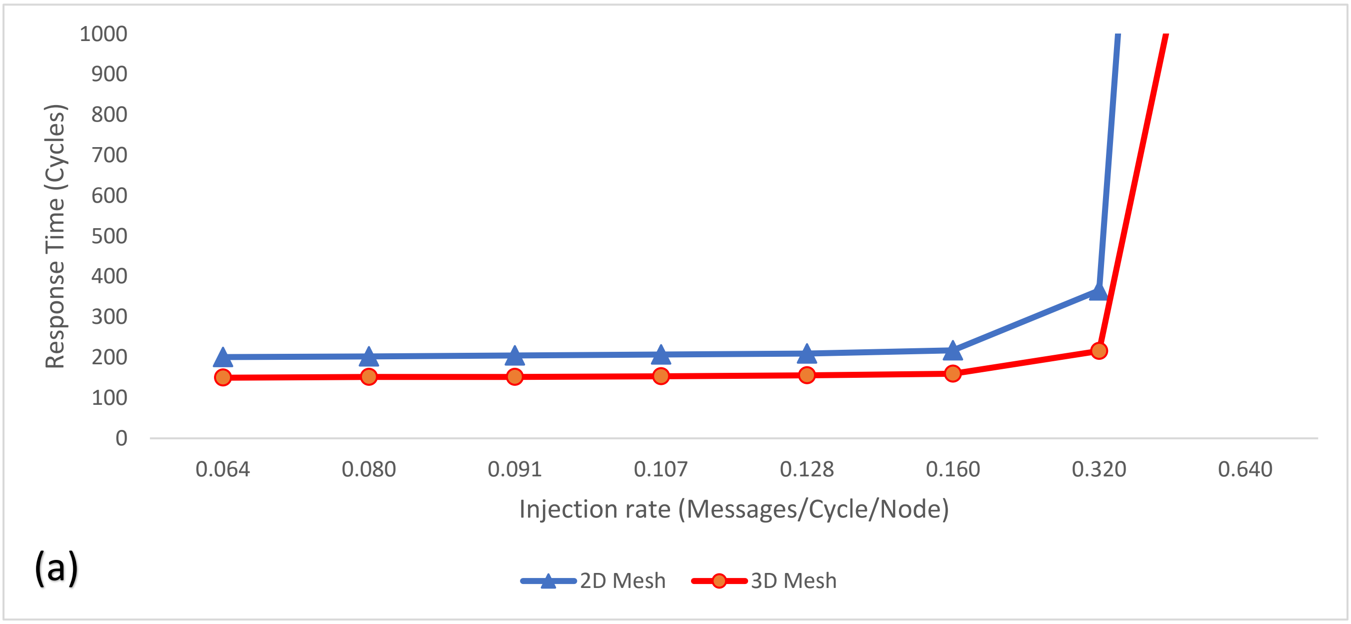
4.6.3 2D vs 3D topologies: Unconstrained implementation

In this comparison the focus is on the impact of increasing the number of dimensions in a network on the overall system performance. This comparison evaluates the 2D versions of each topology against its 3D counterpart for an unconstrained implementation. The purpose of this is to assess the influence of the graph-theoretical properties of 2D topologies on performance when compared to those of their 3D counterparts.

Scenario 1: Uniform traffic

The three topologies are compared once again under the uniform traffic pattern. Figures 4.11 to 4.16 depict the results of the comparison.

The previous figures indicate that the 3D topologies generally outperform their 2D counterparts when subjected to similar traffic loads. This can be attributed to their larger number of paths which allows for a lower average message distance. For instance, in the 3D bidirectional torus with 10x10x10 nodes the average message distance is 7.5 hops whilst in its 2D counterpart with a similar network size (I.e. 32x32) the average message distance is 16 hops. This results in a lower latency for the 3D bidirectional torus compared to its 2D counterpart. The same justifications can be extended to the mesh and unidirectional torus.



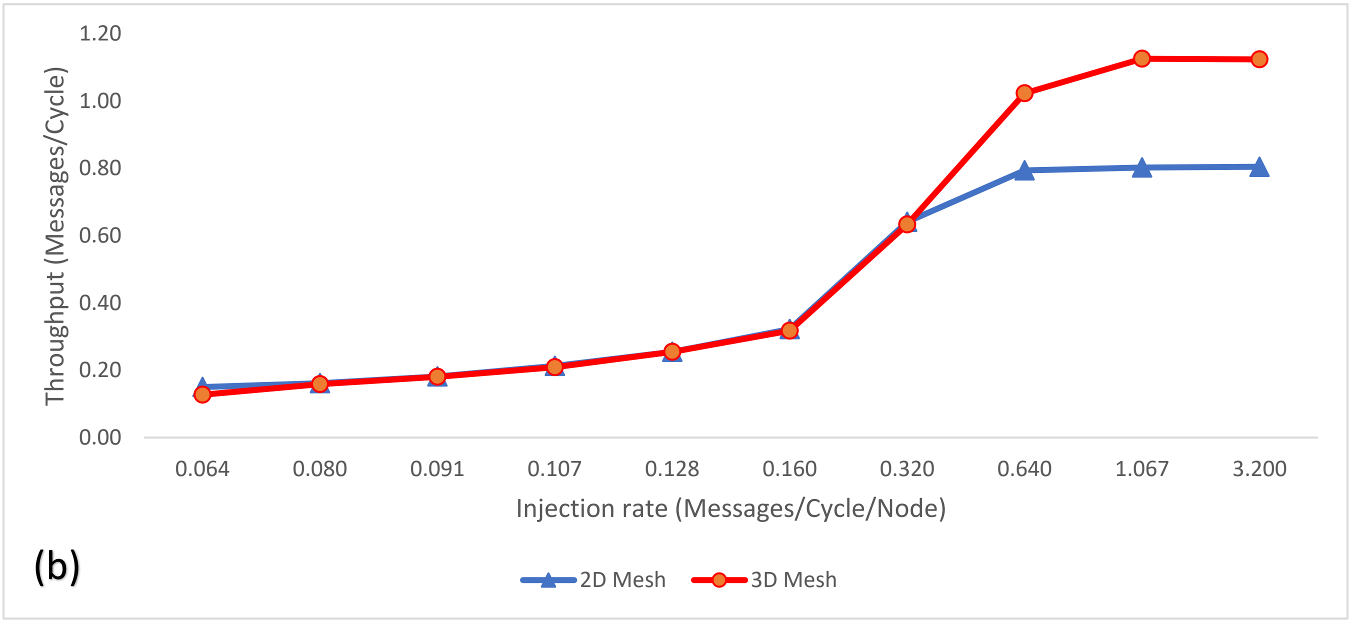
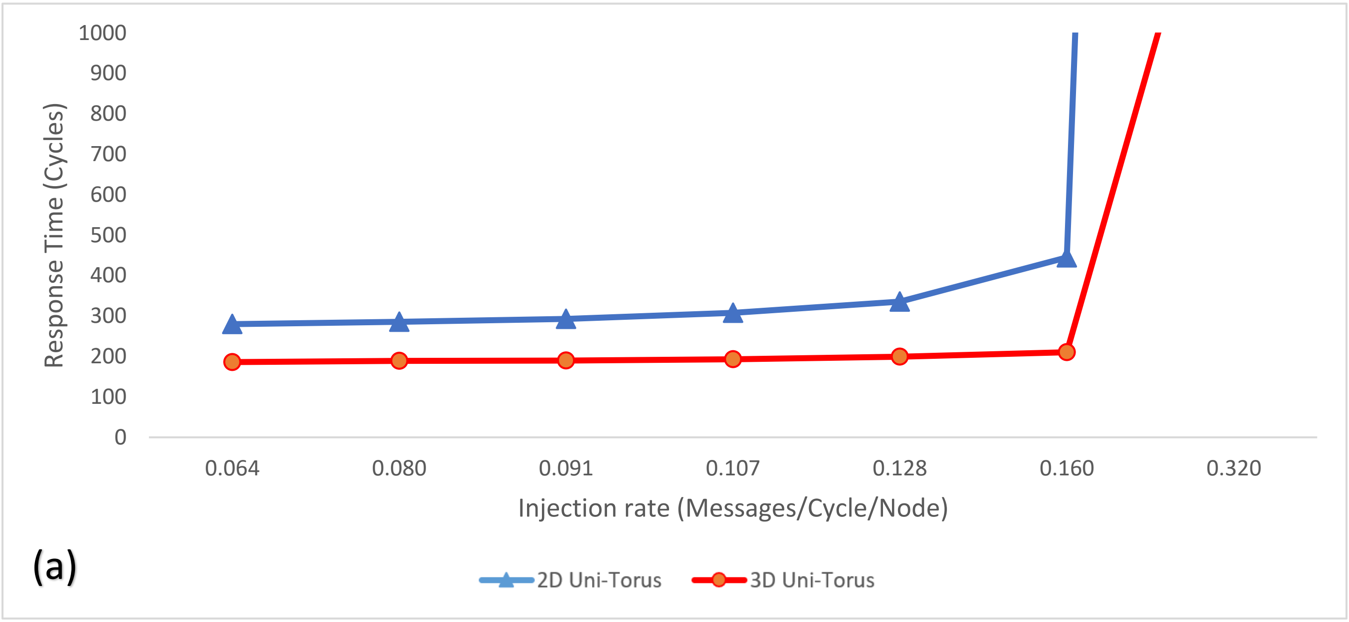


Figure 4.11 Performance results for the 2D vs 3D mesh for network sizes 8x8 vs 4x4x4 nodes under unconstrained uniform traffic. (a) Response time, (b) Throughput.



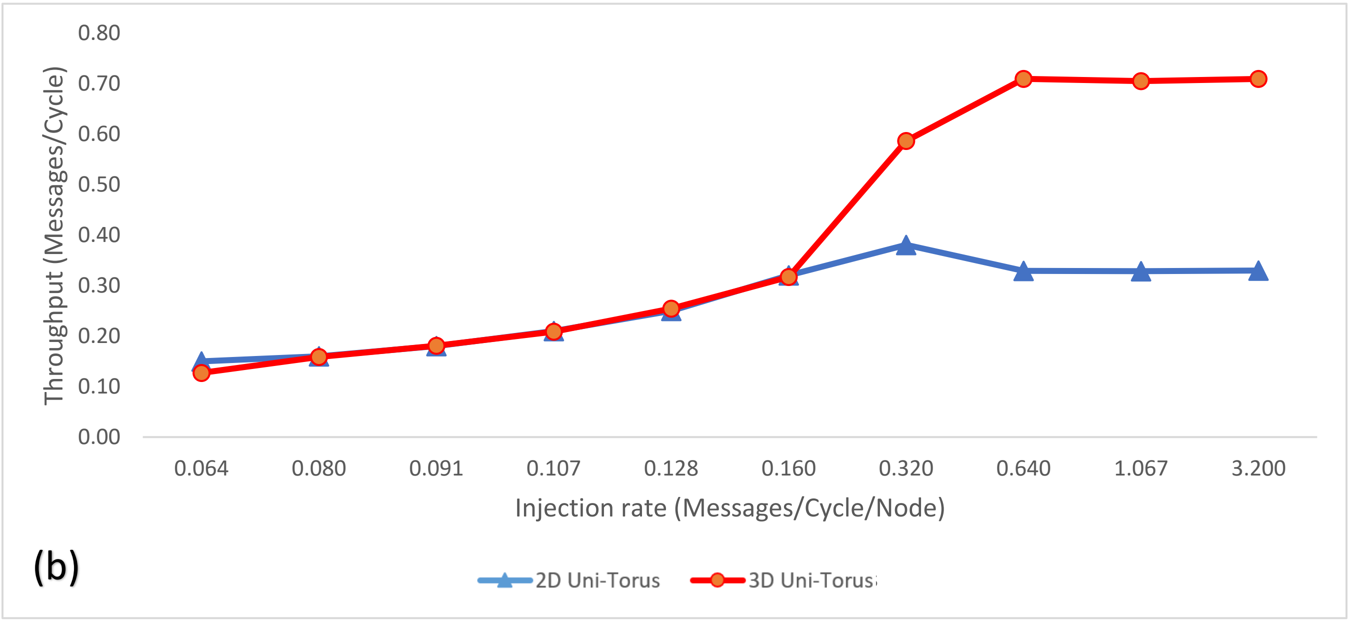
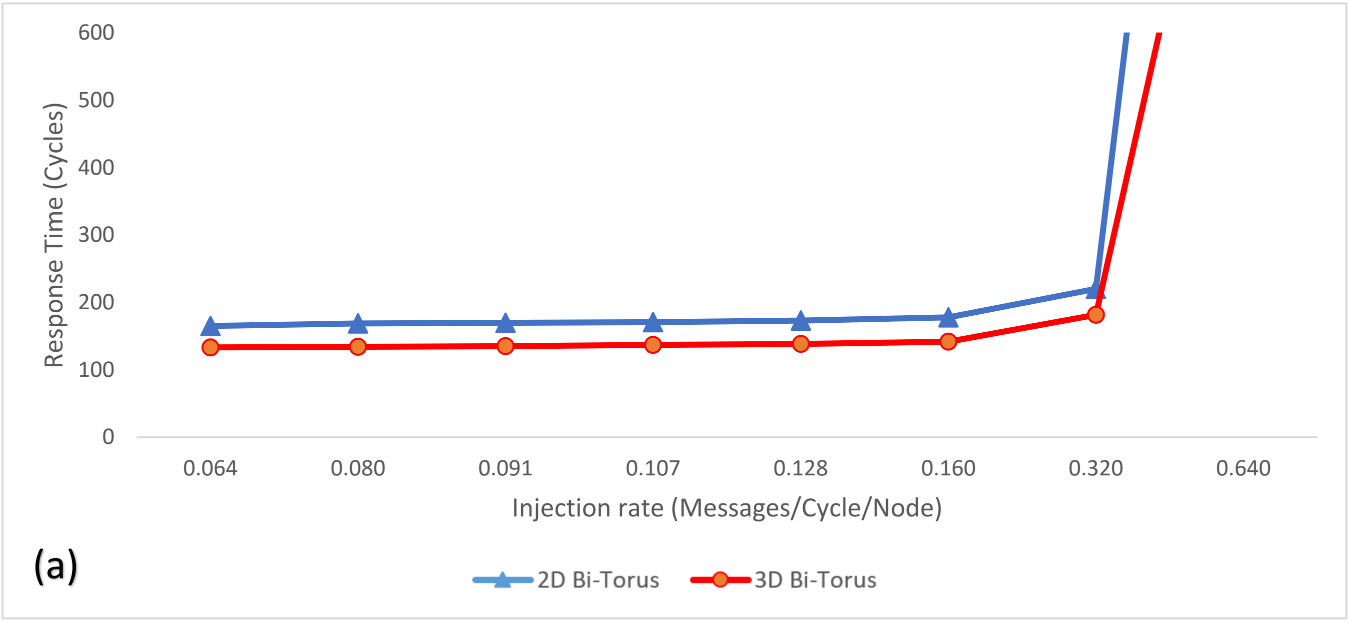


Figure 4.12 Performance results for the 2D vs 3D unidirectional torus for network sizes 8x8 vs 4x4x4 nodes under unconstrained uniform traffic. (a) Response time, (b) Throughput.



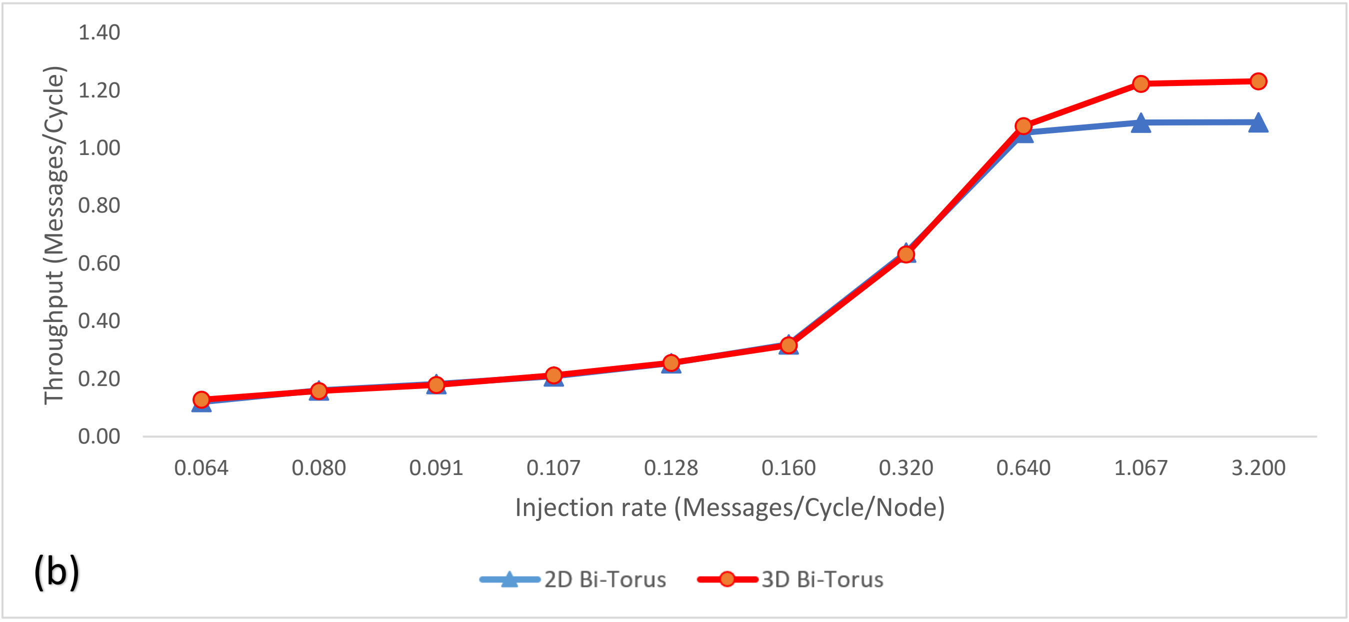
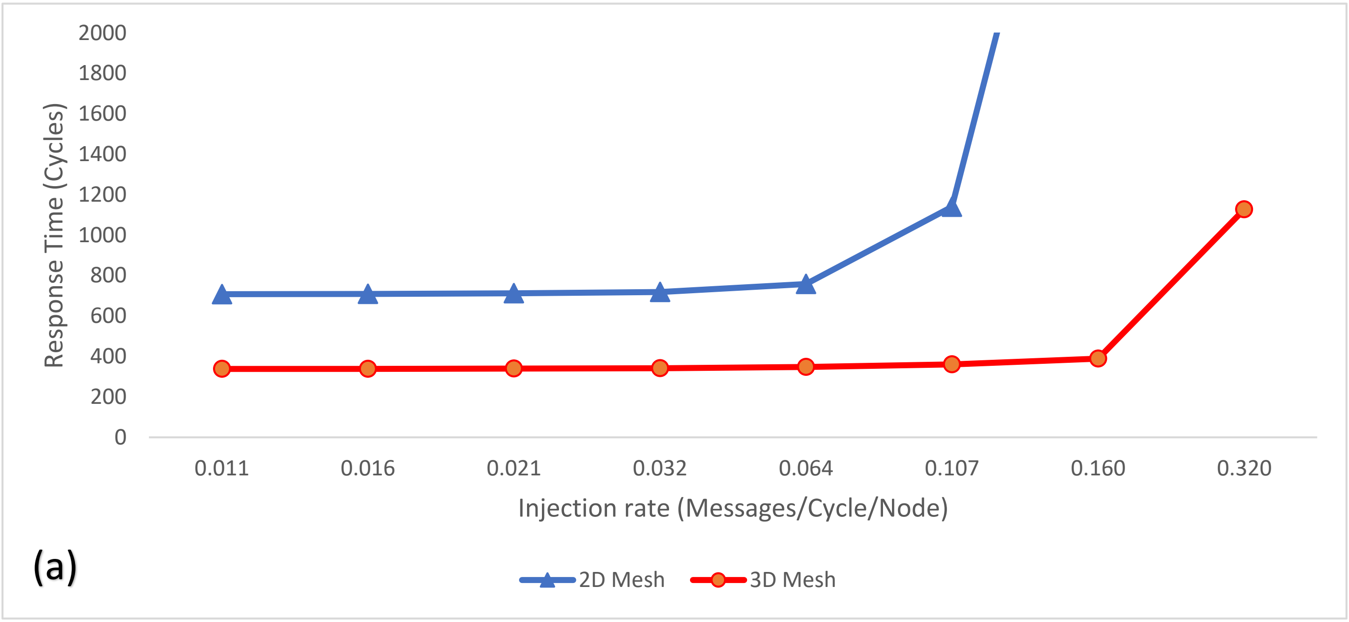


Figure 4.13 Performance results for the 2D vs 3D bidirectional torus for network sizes 8x8 vs 4x4x4 nodes under unconstrained uniform traffic. (a) Response time, (b) Throughput.



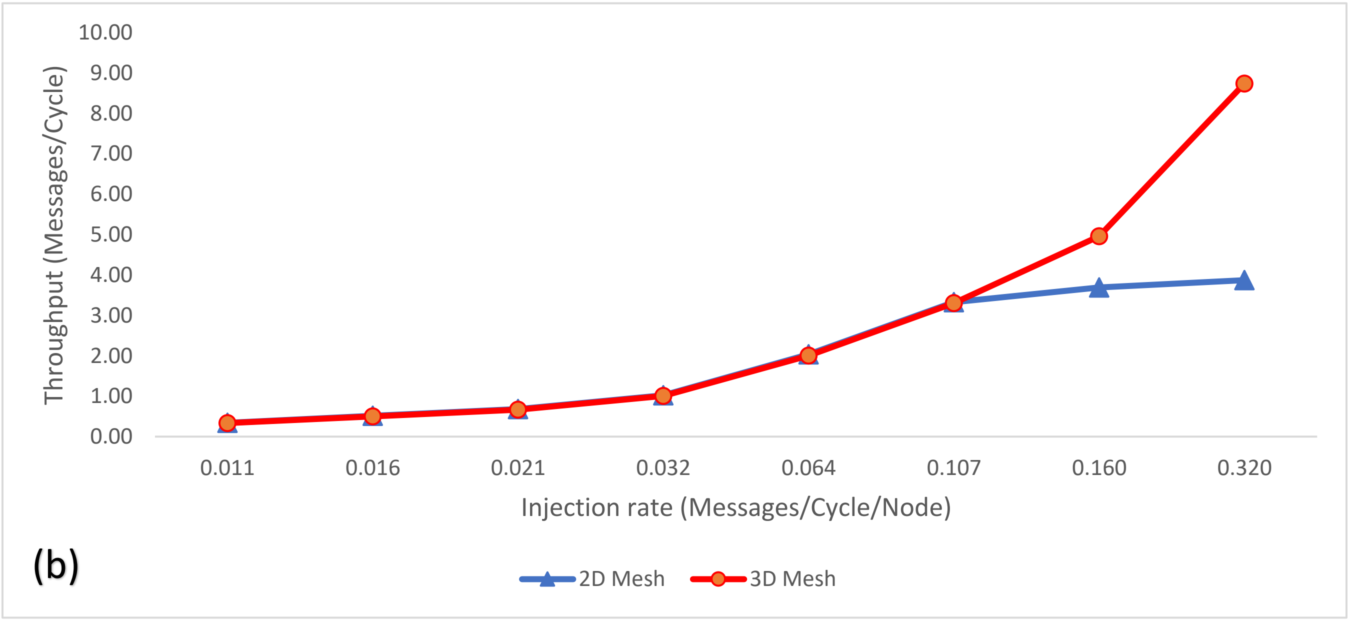
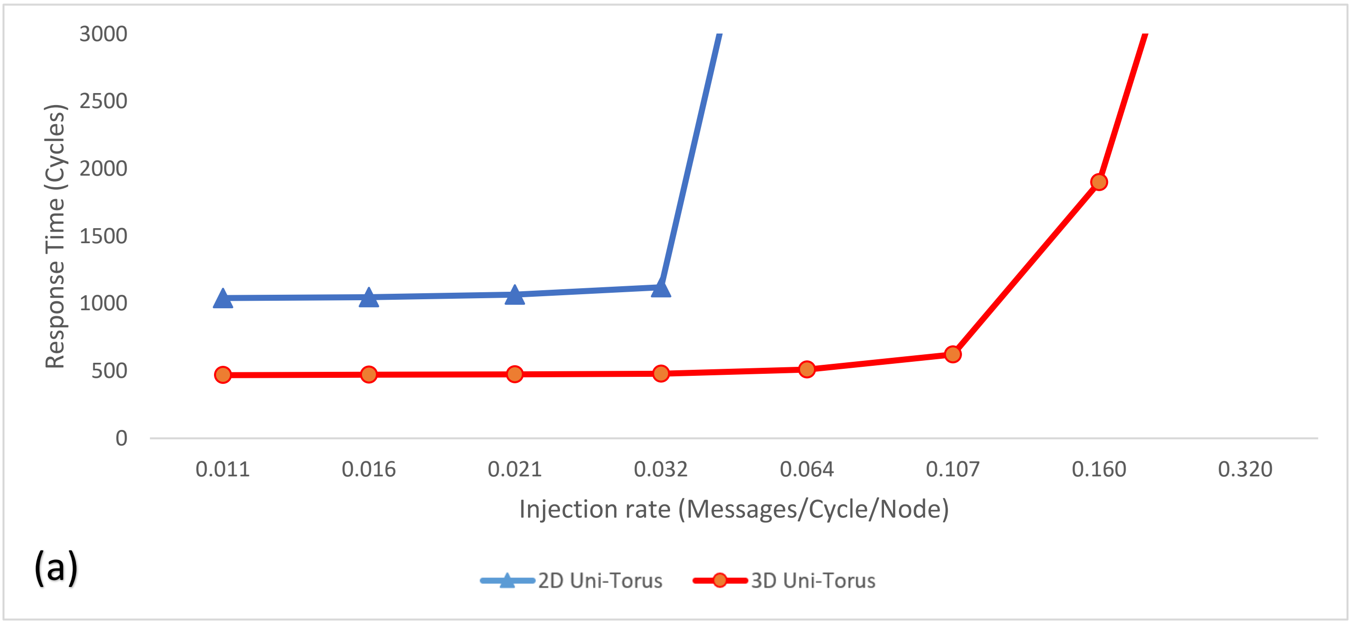


Figure 4.14 Performance results for the 2D vs 3D mesh for network sizes 32x32 vs 10x10x10 nodes under unconstrained uniform traffic. (a) Response time, (b) Throughput.



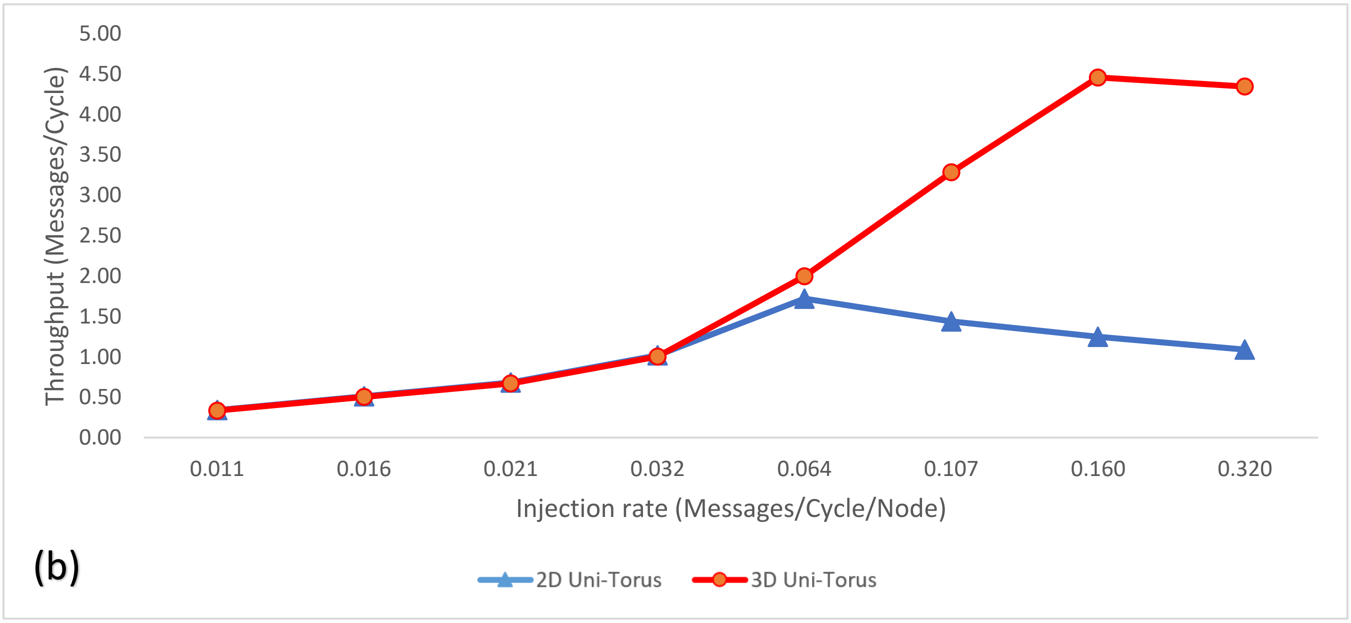
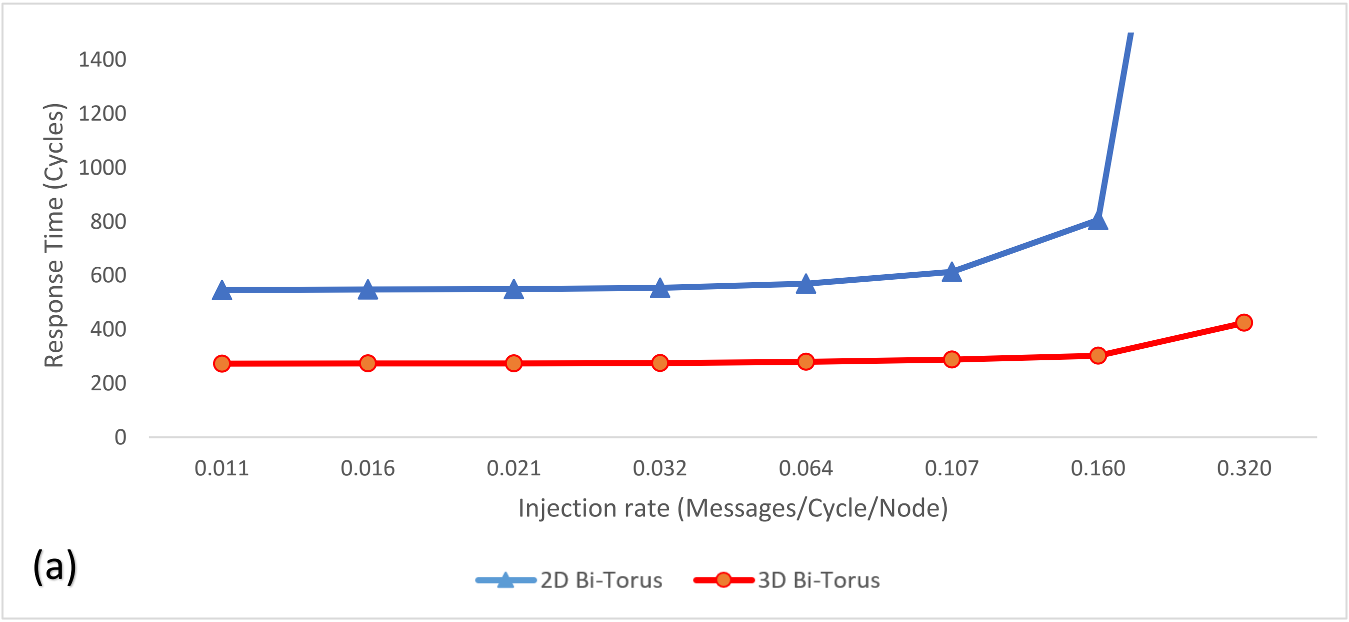


Figure 4.15 Performance results for the 2D vs 3D uni-torus for network sizes 32x32 vs 10x10x10 nodes under unconstrained uniform traffic. (a) Response time, (b) Throughput.



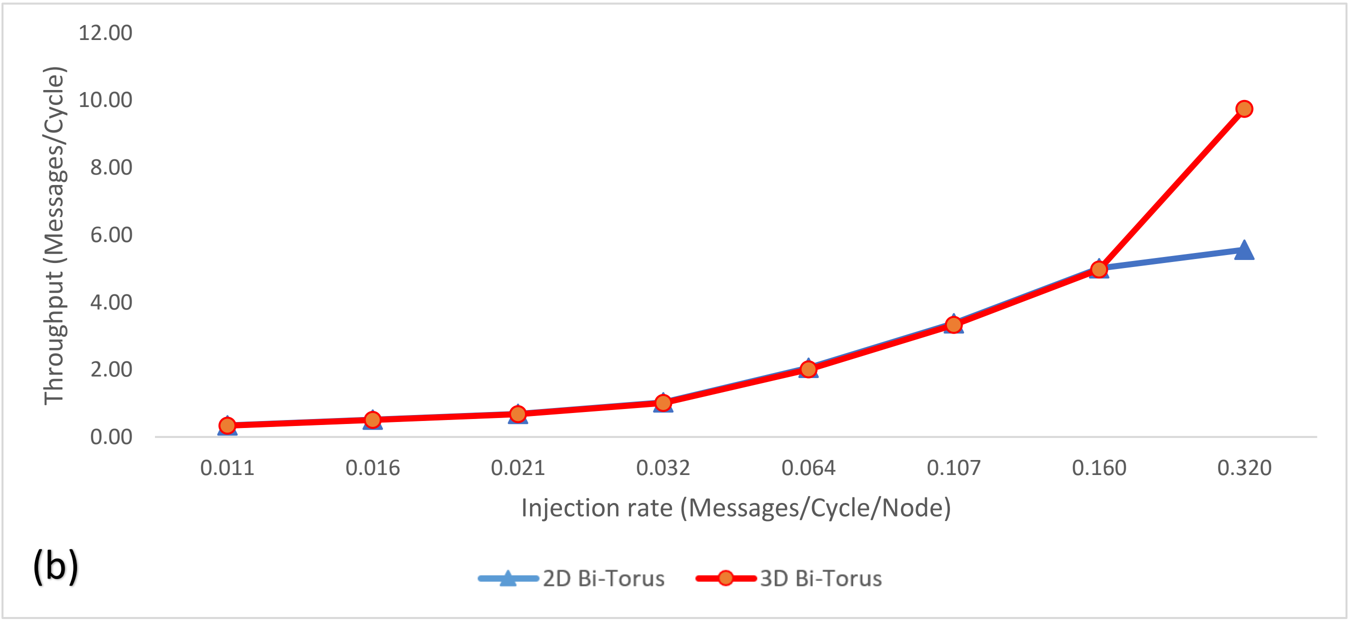


Figure 4.16 Performance results for the 2D vs 3D bi-torus for network sizes 32x32 vs 10x10x10 nodes under unconstrained uniform traffic. (a) Response time, (b) Throughput.

4.6.4 2D vs 3D topologies: Constrained implementation

The purpose of this comparison is to assess the impact of the physical constraints imposed by implementation technology on channel width when attempting to implement a 3D topology on a 2D chip. When a 3D topology is mapped on a 2D plane, the topology has to be stretched, causing more links to cross the center of the plane and thus increasing its bisection width. Each topology will have its 2D version compared against its 3D counterpart for a fixed bisection bandwidth as in Dally's study [35]. Due to space limitations only four distinct network sizes are examined, notably 8x8 versus 4x4x4 and 32x32 versus 10x10x10 to ensure a similar number of nodes across the 2D and 3D topologies.

Assuming a fixed bisection width and using the previously mentioned equations we can compute the channel width in a 3D topology in terms of the channel width of its 2D equivalent. The table below summarizes the channel width for the network sizes examined in our scenarios.

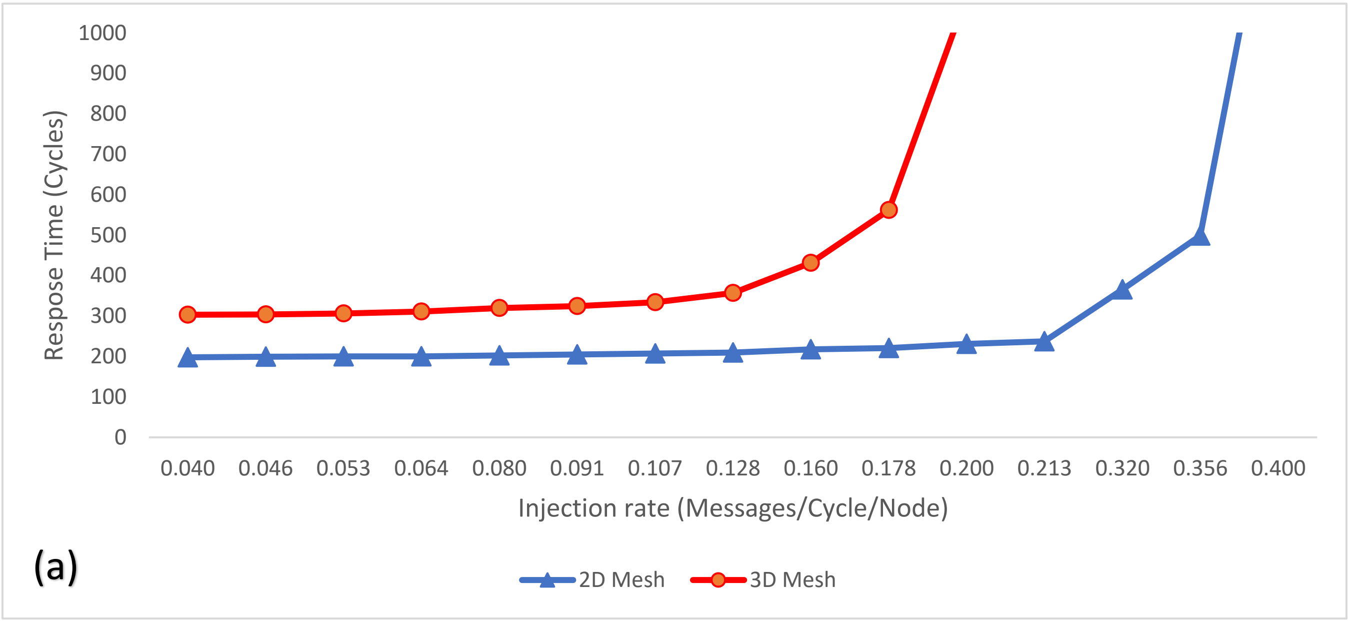
Table 4.2 The ratio of the channel width for 3D topologies to that of 2D topologies

Topology	8x8 versus 4x4x4	32x32 versus 10x10x10
Mesh, unidirectional torus, bidirectional torus	$W_{3D} = \frac{1}{2} W_{2D}$	$W_{3D} = \frac{1}{3} W_{2D}$

Scenario 1: Uniform traffic

The 2D and 3D topologies are assessed under the uniform traffic pattern with physical constraints taken into account.

In figures 4.17 to 4.22, similar conclusions can be drawn from the simulation results for all three topologies. That is, the 2D topologies clearly outperform their 3D counterparts under all traffic loads when it comes to response time. When it comes to the throughput however the 2D and 3D topologies seem to show similar performance under light and moderate traffic, however the difference can be seen under heavy traffic where the 2D topologies have better performance. The conclusion that can be drawn from this is that despite the 3D topologies having superior graph-theoretical properties in terms of average distance, they are not enough to offset the reduction in channel bandwidth caused by physical constraints imposed by implementation technology.



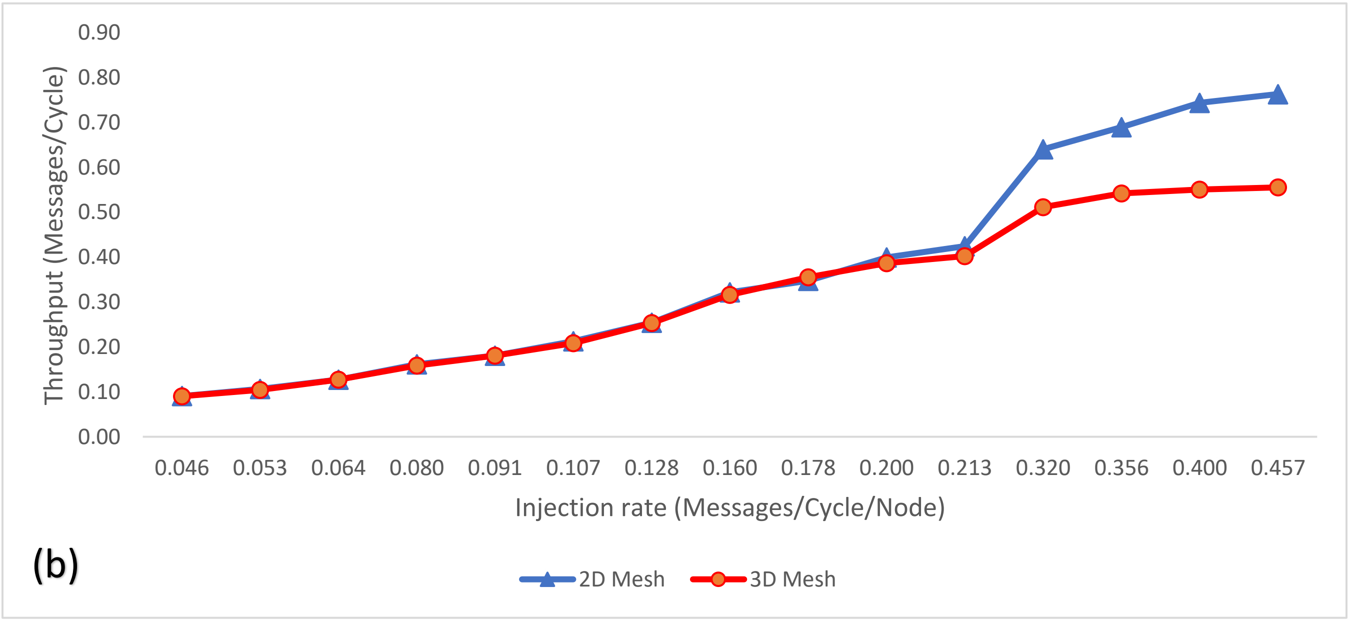
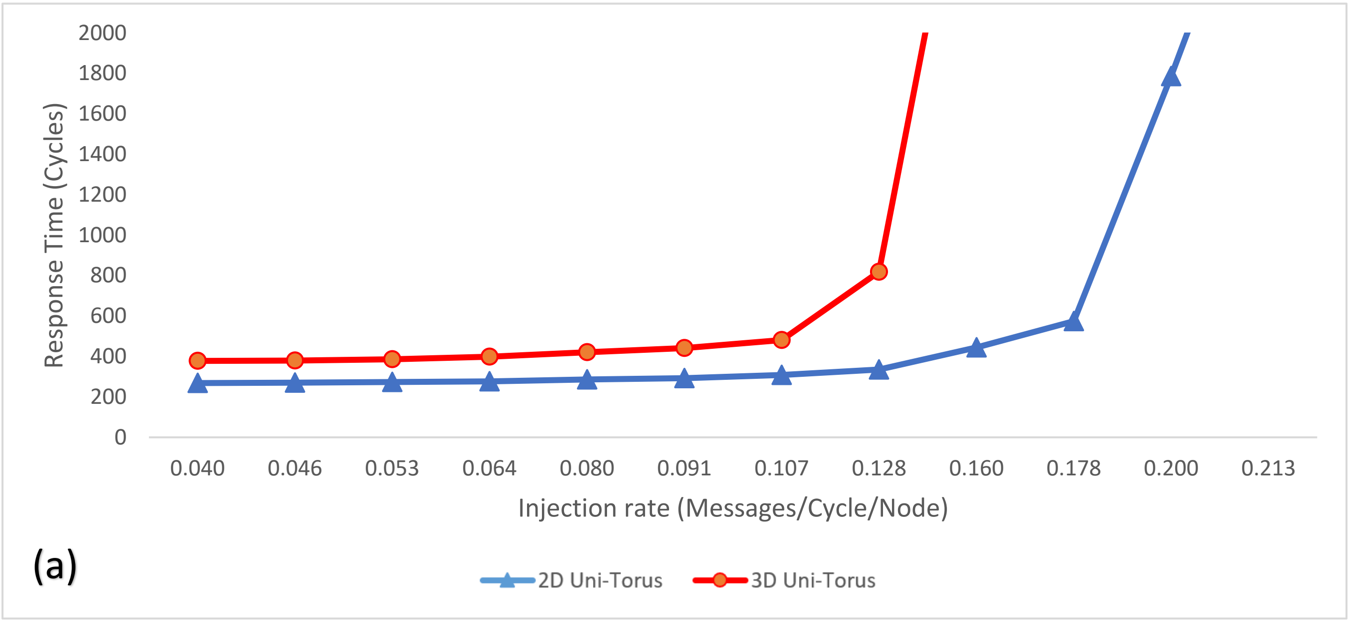


Figure 4.17 Performance results for the 2D vs 3D mesh for network sizes 8x8 vs 4x4x4 nodes under constrained uniform traffic. (a) Response time, (b) Throughput.



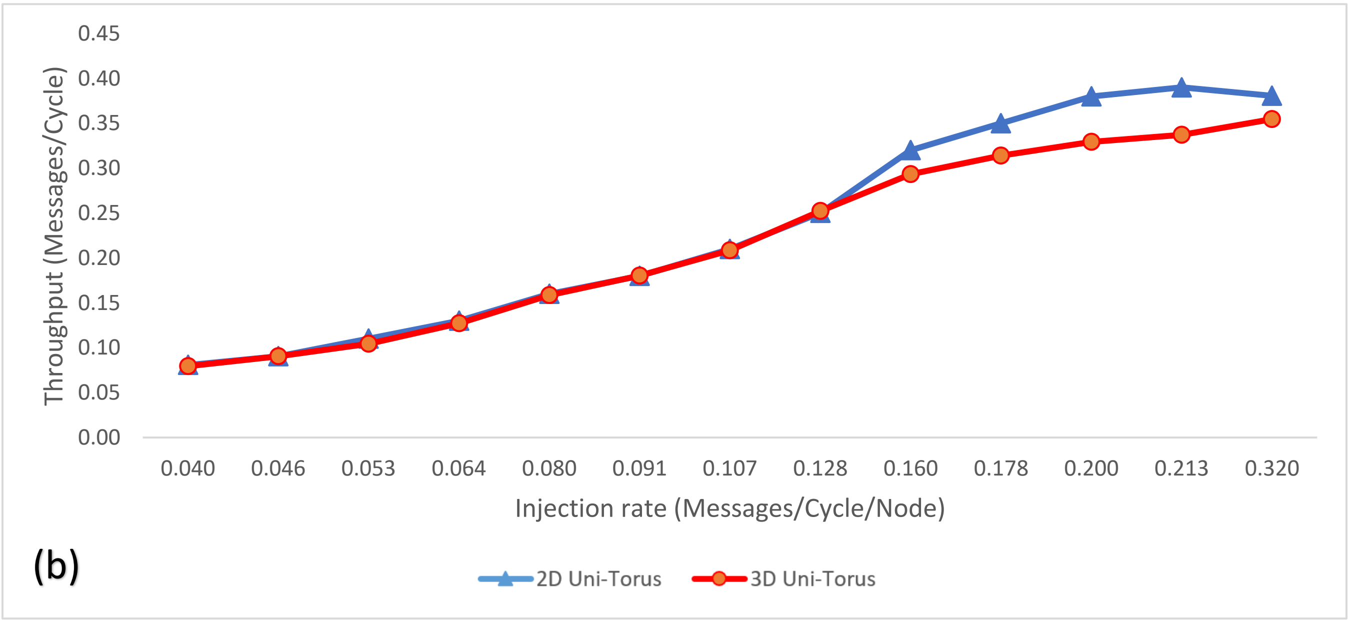
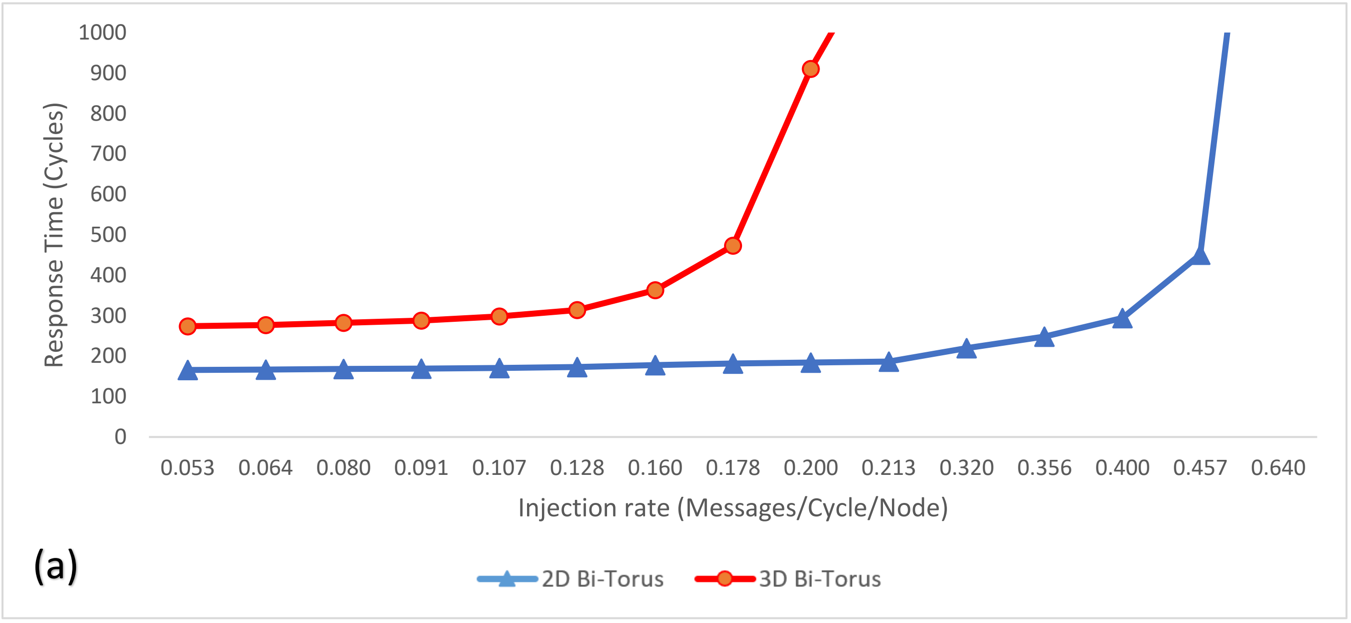


Figure 4.18 Performance results for the 2D vs 3D unidirectional torus for network sizes 8x8 vs 4x4x4 nodes under constrained uniform traffic. (a) Response time, (b) Throughput.



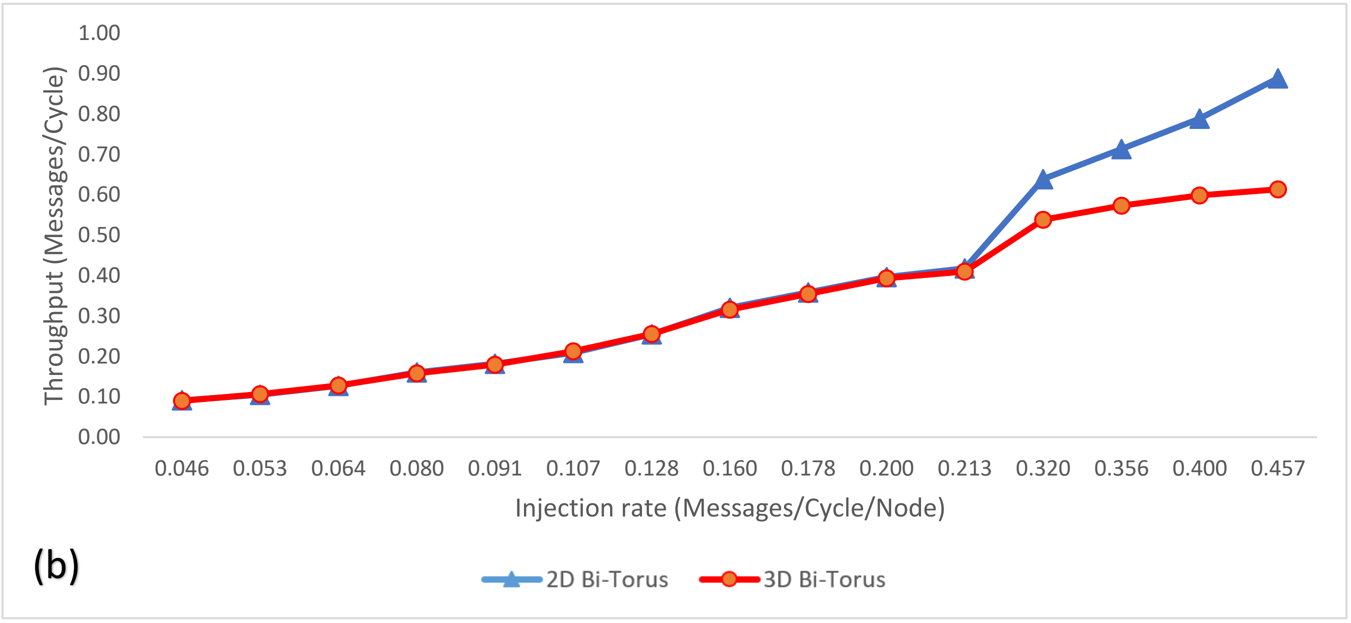
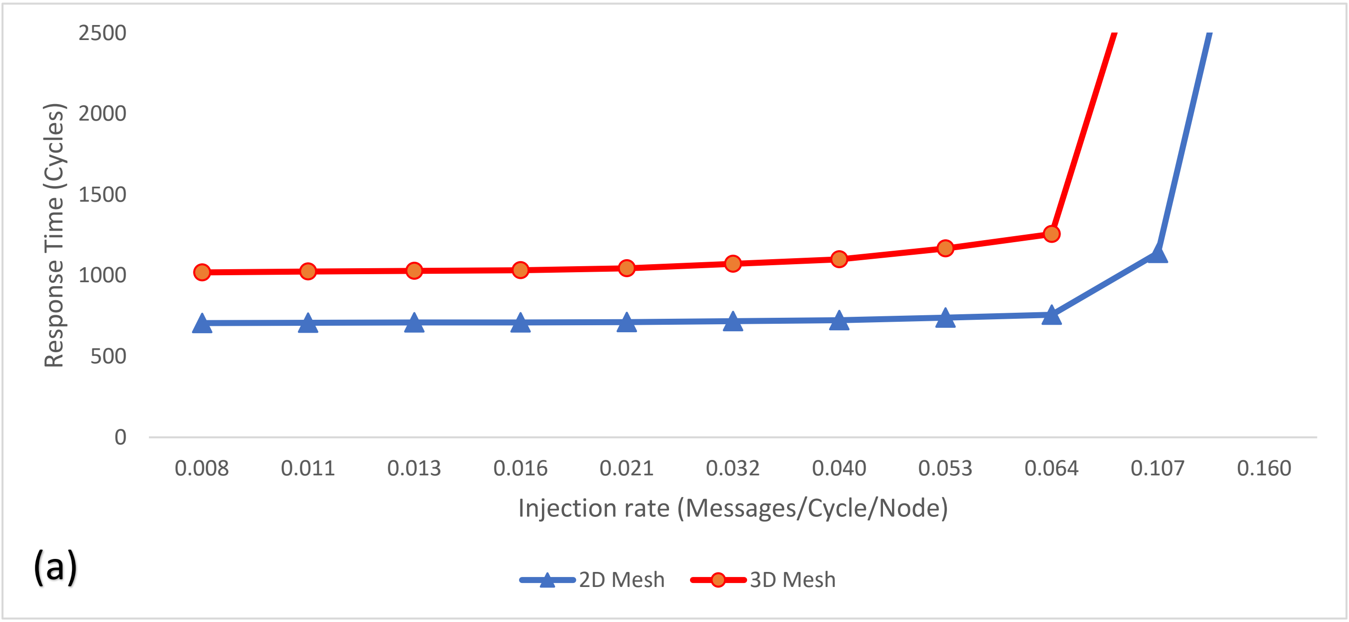


Figure 4.19 Performance results for the 2D vs 3D bidirectional torus for network sizes 8x8 vs 4x4x4 nodes under constrained uniform traffic. (a) Response time, (b) Throughput.



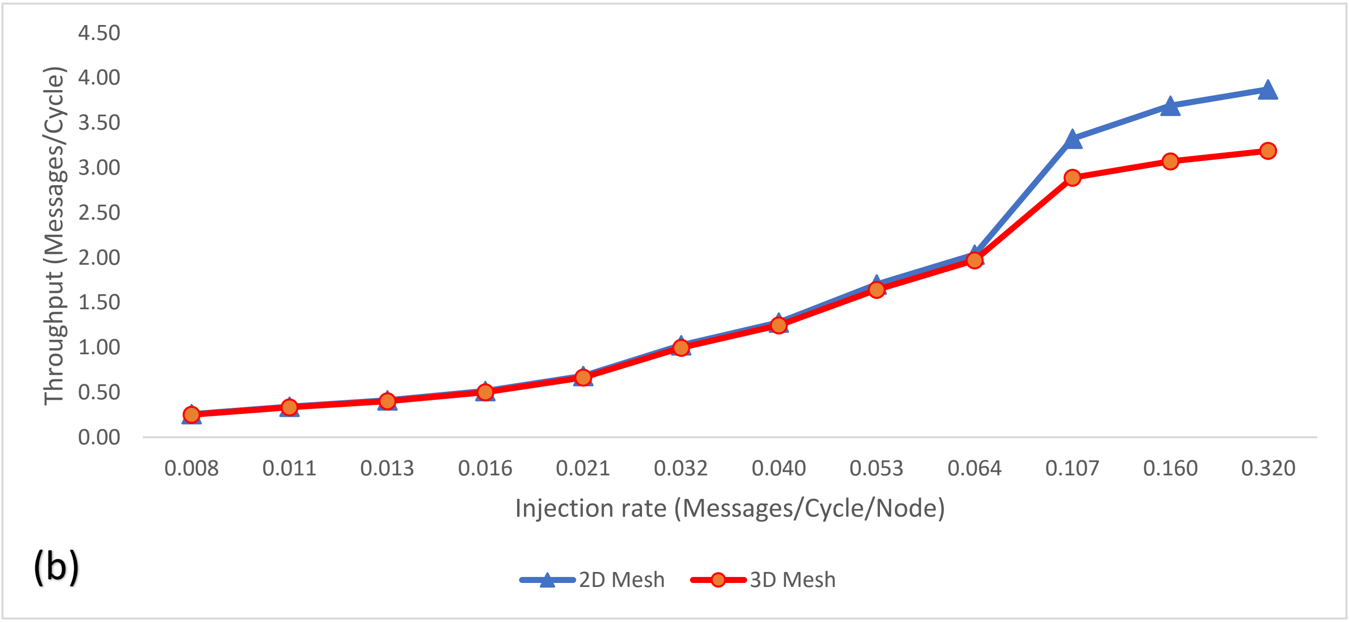
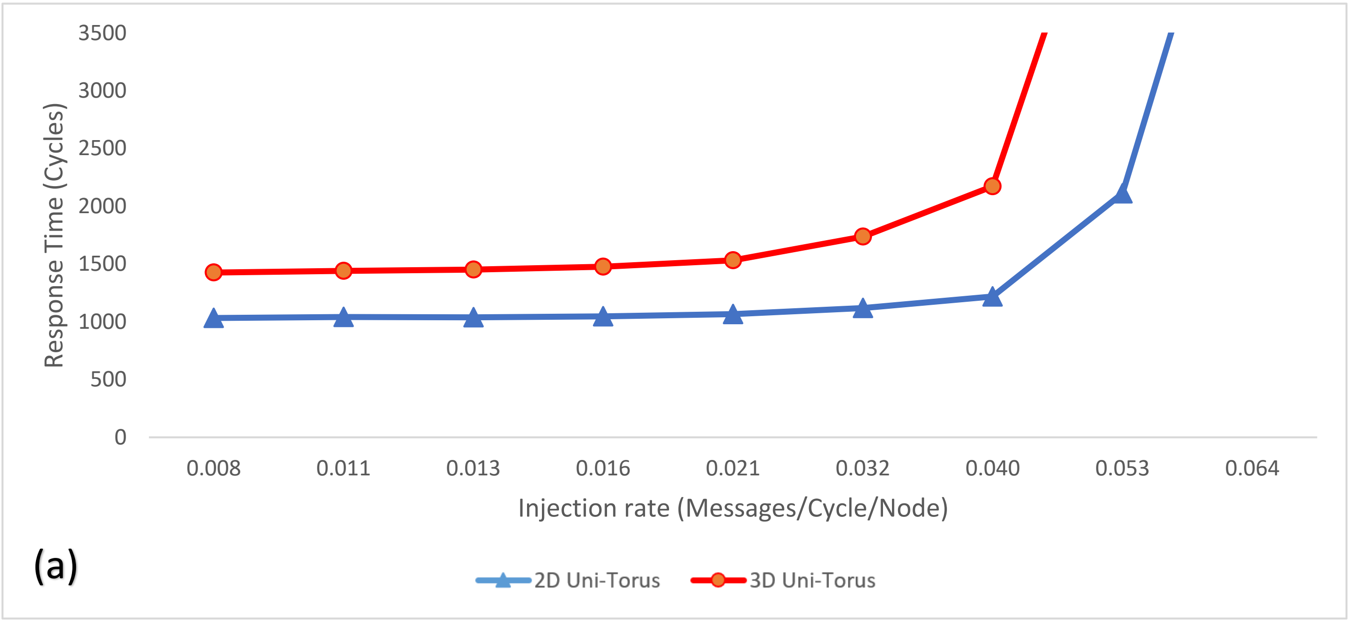


Figure 4.20 Performance results for the 2D vs 3D mesh for network sizes 32x32 vs 10x10x10 nodes under constrained uniform traffic. (a) Response time, (b) Throughput.



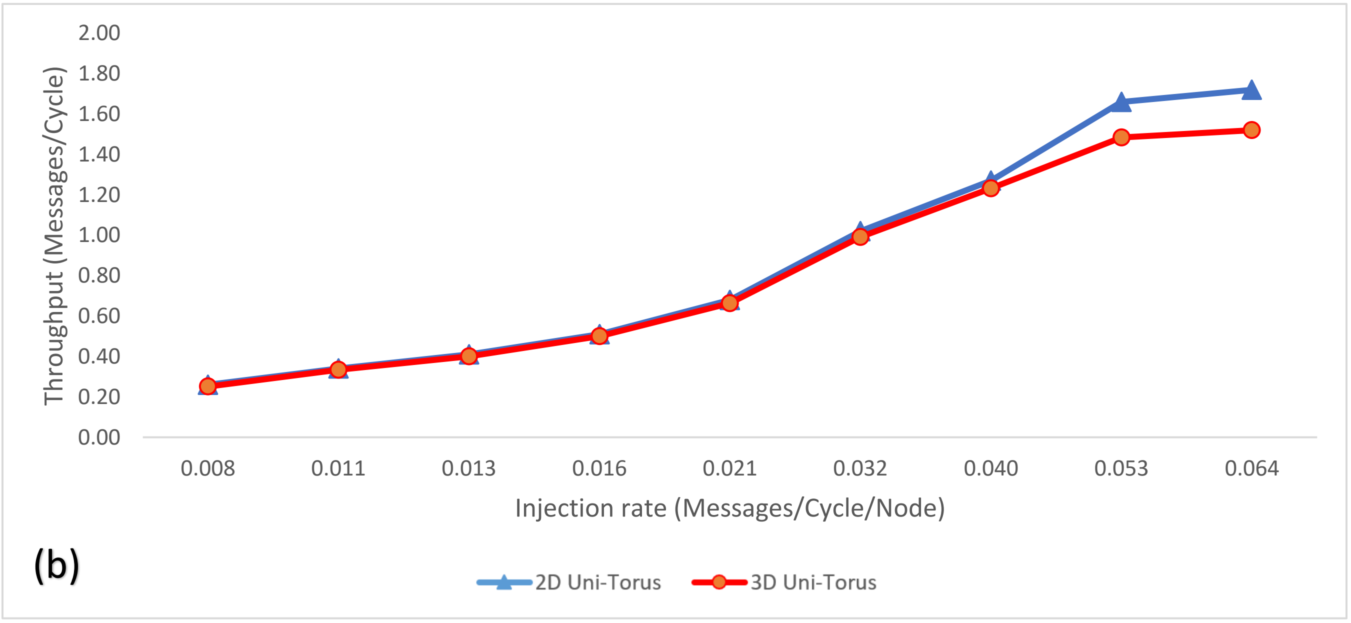
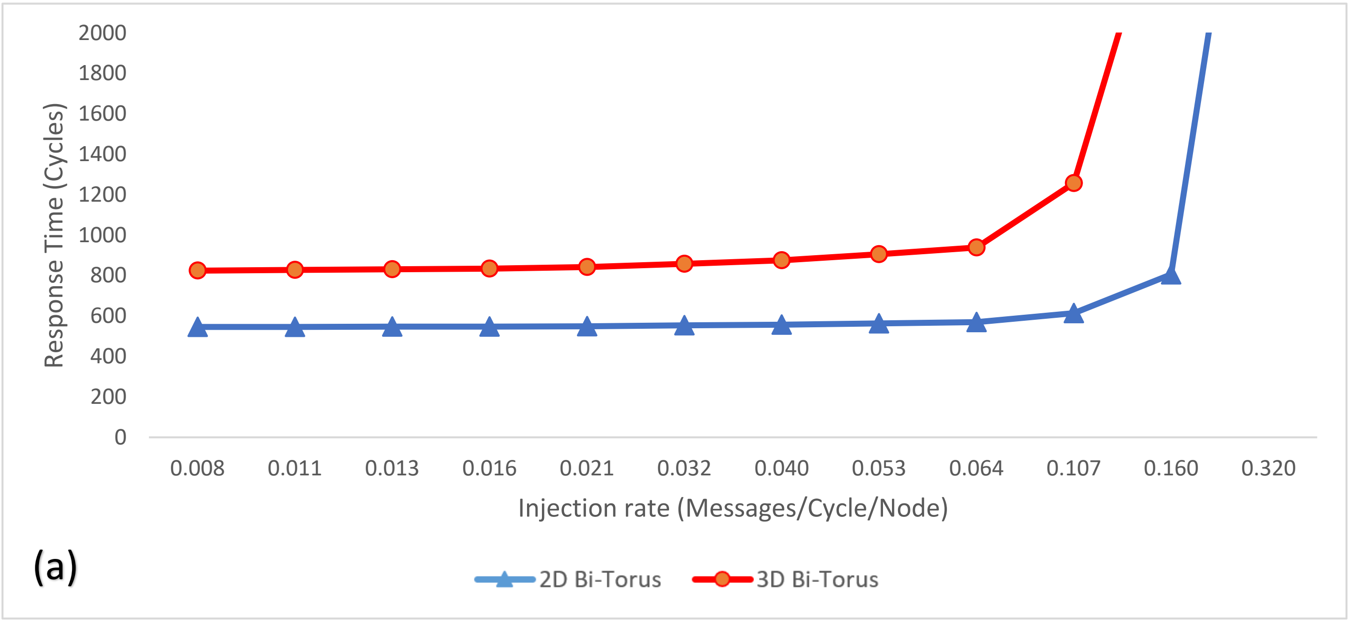


Figure 4.21 Performance results for the 2D vs 3D unidirectional torus for network sizes 32x32 vs 10x10x10 nodes under constrained uniform traffic. (a) Response time, (b) Throughput.



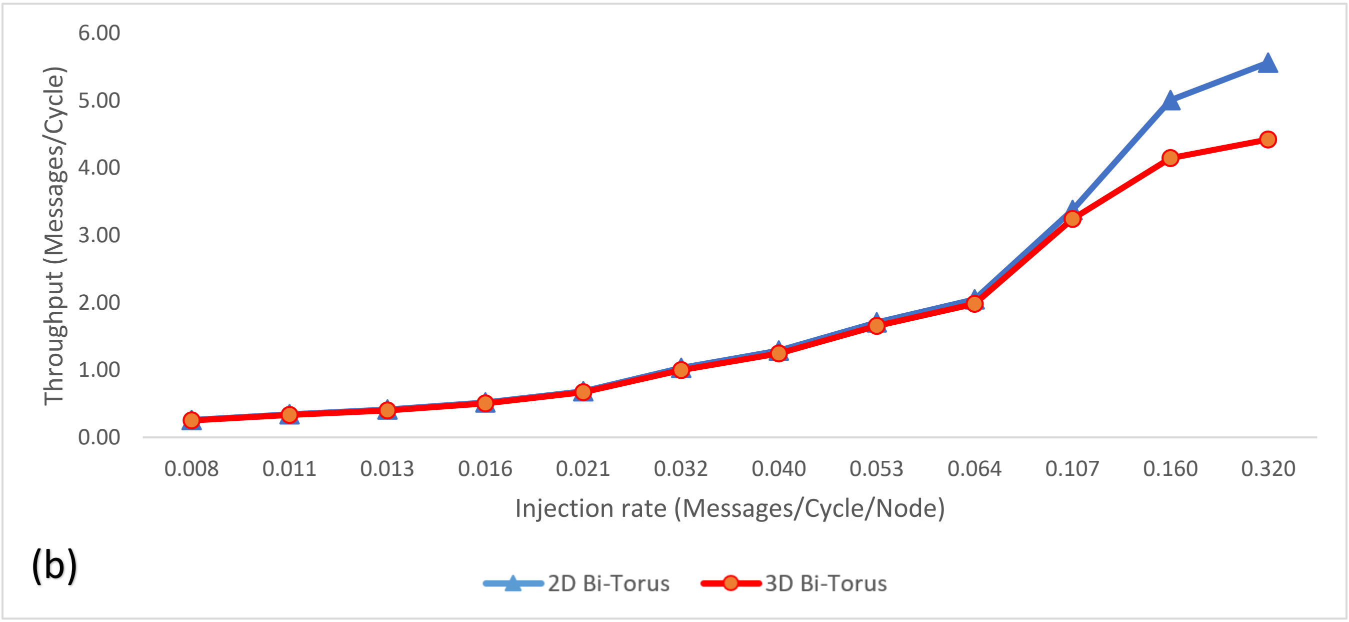


Figure 4.22 Performance results for the 2D vs 3D bidirectional torus for network sizes 32x32 vs 10x10x10 nodes under constrained uniform traffic. (a) Response time, (b) Throughput.

4.7 Conclusions

In order to compare the performance merits of three well-known topologies, namely the mesh, unidirectional, and bidirectional torus, simulation results for the mean response time and throughput have been reported for a number of scenarios. For an unconstrained implementation on channel bandwidth, the bidirectional torus shows the best performance in terms of response time and throughput compared to the others. This is due to the combination of its good average distance along with it being a symmetrical topology. The mesh exhibits performance than the unidirectional torus thanks to its better average distance. In contrast, when implementation constraints on channel bandwidth are taken into consideration the mesh demonstrates superior performance over the unidirectional as well as bidirectional torus. This is mainly due to its higher channel bandwidth manages to offset the detrimental effects of it being an asymmetrical topology and higher average distance in comparison to the bidirectional torus.

Another comparison between the 2D and 3D topologies for the case of the mesh, unidirectional and bidirectional torus has also been carried out. Simulation results have revealed that the 3D topologies deliver better performance in the unconstrained scenarios owing to their superior graph-theoretical properties relative to their 2D counterparts. However once implementation constrains are considered the significant reduction in channel bandwidth for the 3D topologies caused their performance to deteriorate. This allows the 2D versions to exhibit better performance. It is worth mentioning that the observed performance trends are applicable for both uniform and hotspot traffic patterns.

Conclusions and future directions

Much research activities on Systems-on-Chip (SoCs) have gained momentum over the past decades due to the exponential increase in transistor integration into chips as predicted by Moore's Law [1]. This has enabled the implementation of many processing elements inside a single chip. These processing elements are often interconnected by means of routing elements via links forming what is usually known as a Network-on-Chip (NoC).

Numerous research studies have proposed various topologies for NoCs, including the mesh, torus, fat-tree and spidergon. Many of the existing studies have compared the relative performance of these topologies [22, 23, 24, 25, 27, 28]. However, most of these studies have concentrated on the graph-theoretical properties of these topologies and have largely ignored the impact of the constraints imposed by implementation technology on channel bandwidth. The most relevant constraint in the case of NoCs is the wiring density [35]. This is often measured in terms of the bisection width [35].

The aim of our project has been to compare the performance of some well-known topologies notably the mesh and the torus (with its unidirectional and bidirectional variants) while taking into account the implementation constraints. To achieve this, a discrete-event simulation model for these networks has been designed and implemented in C using the Codeblocks IDE. The simulator has been validated using known test cases where the outcomes can be easily predicted.

The simulation model has been used to perform extensive simulation experiments to analyse the performance of the mesh, unidirectional and bidirectional torus under various operating scenarios. When implementation constraints on channel bandwidth are ignored, the simulation results have indicated that the bidirectional torus exhibits the best performance over the mesh and unidirectional torus (for both the 2D and 3D versions) under uniform as well as hotspot traffic patterns. This can be justified by the fact that the bidirectional torus has a lower average distance in comparison to the unidirectional torus and mesh, and having a topology that is symmetric allowing it to distribute the network traffic evenly across its links.

When implementation constraints on channel bandwidth are taken into consideration the mesh and unidirectional torus end up with higher channel bandwidth than the bidirectional torus. The simulation results have revealed that the mesh (for both the 2D and 3D versions) can take advantage of its wider channel bandwidth to mitigate the negative effects of its asymmetrical topology and higher average distance in comparison to the bidirectional torus. The unidirectional torus, however, does not manage to exploit its higher channel bandwidth to compensate for its higher average distance. In other words, the mesh exhibits lower response times and higher throughput when subjected to uniform and hotspot traffic patterns.

Another comparison between the 2D and 3D topologies for the case of the mesh, unidirectional and bidirectional torus has also been carried out. Simulation results have shown that the 3D topologies deliver better performance in the unconstrained scenarios. This has been attributed to their larger number of paths which in turn allows for a significantly lower average distance relative to their 2D counterparts.

When implementation constraints are imposed on channel bandwidth the performance of the 3D topologies worsens significantly compared to the 2D versions. This is because of the large reduction in channel bandwidth as a result of fixing the bisection width. This allows the 2D versions to exhibit better performance. It is worth mentioning that the observed performance trends are applicable for both uniform and hotspot traffic patterns.

There are a number of possible directions that can be pursued in order to further extend our work and these are listed below.

If the necessary computing resources were available it would be interesting to run simulations for large network sizes (e.g. thousands of nodes). This is motivated by Moore's Law that predicts that NoCs with thousands of nodes would be a reality in the foreseeable future.

Many adaptive routing algorithms have been proposed in the literature [2] which can take advantage of the various paths that exist in a given topology to improve network performance. A possible extension of this work would be to extend our simulation model to incorporate adaptive routing and evaluate its influence on the performance properties of the mesh and torus networks.

A popular alternative to packet switching is virtual cut-through as it enables the reduction of response time under light to moderate traffic by avoiding the necessary buffering at intermediate routing elements. It would be interesting to adapt our simulation model to include this switching technique and quantify its influence on the outcome of any comparative study of competing NoC topologies.

Applications typically exhibit various communication patterns between the processing elements including broadcast and multicast. A natural extension of our work would be to develop the simulation model further to accommodate these traffic patterns and assess their impact on the performance of NoC topologies.

References

[1] Konstantinos Tatas, Kostas Siozios, Dimitrios Soudris and Axel Jantsch, "Designing 2D and 3D Network-on-Chip Architectures," *Springer Science+Business,* 2014.

[2] Jose Duato, Sudhakar Yalamanchili and Lionel M. Ni, "Interconnection Networks - An Engineering Approach," *Elsevier Science (USA),* 2003.

[3] Rajeev Kamal, Pankaj Goyal and Vikas Nehra, "Network on Chip: Topologies, Routing, Implementation," *International Journal of Advances in Science and Technology,* vol. 4, no. 1, pp. 24-34, January 2012.

[4] Mehdi Baboli, Nasir Shaikh Husin and Muhammad Nadzir Marsono, "A Comprehensive Evaluation of Direct and Indirect Network-On-Chip Topologies," *International Conference on Industrial Engineering and Operations Management ,* January 2014.

[5] Salem Umamaheswari, Raja Paul Perinbam, K. Monisha and Jahir Ali, "Comparing the Performance Parameters of Network on Chip with Regular and Irregular Topologies," *Springer-Verlag Berlin Heidelberg,* pp. 177-186, 2011.

[6] Vikram S. Adve and Mary K. Vernon, "Performance analysis of mesh interconnection networks with deterministic routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 3, pp. 225–246, March 1994.

[7] Cruz Izu, "A router node architecture for cut-through torus networks." *Technical Report, Departamento deArquitectura y Tecnologia de Computadores*,Universidad del Pais Vasco, Spain, 1994.

[8] Lei Yang, Weichen Liu, Weiwen Jiang, Mengquan Li, Peng Chen and Edwin Hsing-Mean Sha, "FoToNoC: A Folded Torus-Like Network-on-Chip Based Many-Core Systems-on-Chip," *IEEE Transactions on Parallel and Distributed Systems,* vol. 28, no. 7, pp. 1905-1918, December 2016.

[9] Ali Rezaei Aliabad, "Tree Topology," *Int. J. Contemp. Math. Sciences*, vol. 5, no. 21, pp. 1045-1054, January 2010.

[10] Arpit Jain, Alok Kumar Gahlot, Rakesh Dwivedi, Adesh Kumar and Sanjeev Kumar Sharma, "Fat Tree NoC Design and Synthesis," pp. 1749-1756, January 2018.

[11] John Kim, James Balfour and William J. Dally, "Flattened Butterfly Topology for On-Chip Networks," *IEEE Computer Architecture Letters,* March 2007.

[12] Suyog K. Dahule, Pallavi D. Tiware and Sagar Soitk, "Review on Network on Chip (NoC) Topology," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 4, no. 5, May 2016.

[13] Jie Chen and Cheng Li, "Network-on-Chip (NoC) Topologies and Performance: A Review," 2011.

[14] Andrew S. Tanenbaum, "Computer networks", *Prentice-Hall Int., Inc*. (2nd Ed.), 1989.

[15] Suyog K. Dahule, Sagar Soitkar and Vaishali Ingle, "A Review Paper on Different Switching Techniques in NOC Router Architecture," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 4, no. 11, pp. 227-229, November 2014.

[16] Parviz Kermani and Leonard Kleinrock, "Virtual cut-through: A new computer communication switching technique", *Comp. Networks*, Vol. 3, pp 267-286, 79.

[17] Charles L. Seitz, "The hypercube communication chip", Dep. Comp. Sci., CalTech, Display File 5182:DF:85, March 85.

[18] Ville Rantala, Teijo Lehtonen and Juha Plosila, "Network on Chip Routing Algorithms," *Turku Center for Computer Science,* August 2006.

[19] Patrick T. Gaughan, "Adaptive routing protocols for hypercube interconnection networks", *IEEE Computer*, May 93.

[20] Seth Abraham, "Issues in the architecture of direct interconnection schemes for multiprocessors", Ph.D. thesis, Univ. of Illinois at Urbana-Champaign, 90.

[21] Manoj Kumar, Vijay Laxmi, Manoj Singh Gaur, Masoud Daneshtalab, Pankaj, Seok-Bum Ko and Mark Zwolinski, "A Novel Non-minimal/Minimal Turn Model for Highly Adaptive Routing in 2D NoCs," *IEEE Computer*, 2014.

[22] Niranjan Chiplunkar, Muhammed Khalid, V. Sanju, Sujata Joshi and Jaya. S. Nirmala, "A Performance Study of 2D Mesh & Torus for Network on Chip Based System," *Elsevier Publications*, vol. 1, 2013.

[23] Mohammed Mirza Aghatabar, Somayyeh Koohi, Shaahin Hessabi and Massoud Pedram, "An Empirical Investigation of Mesh and Torus NoC Topologies Under Different Routing Algorithms and Traffic Models," *IEEE Computer Society*, 2007.

[24] Wu Ning, Ge Fen and Wang Qi, "Simulation and Performance Analysis of Network on Chip Architectures Using OPNET," *IEEE Computer Society*, 2007.

[25] Luciano Bononi and Nicola Concer, "Simulation and Analysis of Network on Chip Architectures: Ring, Spidergon and 2D Mesh," *IEEE Transactions on*, 2006.

[26] International Journal of Computer Applications, "A Comparative Study of Different Topologies for Network-On-Chip Architecture," *Computer Networks,* 2013.

[27] Meaad Fadhel, Ali Qasem and Huaxi Gu, "Square-Octagon Interconnection Architecture for Network -on- Chips," *Network IEEE*, 2014.

[28] Alexander Yin, Nan Chen, Pasi Liljeberg and Hannu Tenhunen, "Comparison of Mesh and Honeycomb Network-on-Chip Architectures," IEEE Computer Society, 2011.

[29] Raj Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling," *WILEY PROFESSIONAL COMPUTING,* April 1991.

[30] Prateek Sharma, "Discrete-Event Simulation," *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH,* vol. 4, no. 4, pp. 136-140, April 2015.

[31] Adekitan Aderibigbe, "MONTE CARLO SIMULATION," *Proceedings of the 2008 Winter Simulation Conference,* September 2014.

[32] Nejib Mediouni, Salem Hasnaoui, Samir Ben Abid and Oussama Kallel, "Modeling and Performance Evaluation of 2D and 3D NoCs using Discrete Event Simulation," *International Journal of Computer Applications,* vol. 137, no. 12, March 2016.

[33] Gopan S. Sangeetha, Vignesh Radhakrishnan, Prabhu Prasad, Khyamling Parane and Basavaraj Talawar, "Trace-Driven Simulation and Design Space Exploration of Network-on-Chip Topologies," *IEEE/ACM Transactions on Networking,* December 2018.

[34] Khaled Salah Mohamed, "Work Around Moore’s Law: Current and Next Generation Technologies," *Applied Mechanics and Materials,* Vols. 110-116, pp. 3278-3283, 2012.

[35] William J. Dally, Performance analysis of k-ary n-cubes interconnection networks, IEEE Trans. Comp., Vol. 39, No. 6, June 90.

[36] Kajal Agrawal, Milind Shah and Gaurav Asari, "A Review Paper on Multiplier Algorithms for VLSI Technology," *International Journal of Scientific Research in Science, Engineering and Technology,* vol. 4, no. 2, pp. 205-209, January 2018.