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 fulfilment of the Master degree
Domain: MI
Affiliation: Informatics
Specialisation: Computer systems and networks

Ziraoui Zakaria

Jury:
President: Mrs. Aroussi Sanaa
Examiner: Mr. Benyahia Mohamed

Supervisor:
Mr. Ould-Khaoua Mohamed

Academic year: 2020/2021

Acknowledgment

For the first and foremost, I would like to express my deepest gratitude for Almighty God, for his marvellous and amazing grace, for the countless blessings and love so I had finally completed this thesis.

I am over helmed in all humbleness and gratefulness to acknowledge my depth to all those who have helped me to put these ideas, well above the level of simplicity and into something concrete.

I would like to express my special thanks of gratitude to my supervisor, Prof. Mohamed Ould-Khaoua, whose sincerity, encouragement and wisdom I will never forget.
Prof. Mohamed Ould-Khaoua has been an inspiration as I hurdled through the path of this Master degree. He is the true definition of a leader and the ultimate role model. This thesis would not have been possible without him, whose guidance from the initial step in research enabled me to develop an understanding of the subject. I am thankful for the extraordinary experiences he arranged for me and for providing opportunities for me to grow professionally and personally. It is a great honour to learn from Prof. Mohamed Ould-Khaoua.

I am grateful for my parents whose constant love and support keep me motivated and confident. My accomplishments and success are because they believed in me, reminding me of what is important in life, and they are always supportive of my adventures. I am forever thankful for the unconditional love and support throughout the entire thesis process and every day.

My sincere thanks also go to the members of the jury for their interest in my project by agreeing to examine the work and enrich it with their suggestions and recommendations. I am extremely grateful to all my professors and instructors who taught me during my Licence and Master degrees and who have made me what I'm today.

Finally, I 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 conduct a performance evaluation of some well-known topologies that have been proposed for Networks-on-Chip (NoC) including the Mesh and Express Cube along with its variations. This later is based on a simple Mesh interconnection network augmented by long-hop wires called *express channels* to enable bypassing of intermediate hops for non-local communication. Whereas existing studies have focused on the graph-theoretical merits of such topologies, our present study examines the performance of networks-on-chip taking into account the constraints imposed by implementation technology. The most relevant constraint for NoC systems 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 have then been analysed using statistical methods. Our results reveal that while the Express Cube has superior performance when technological constraints are ignored due to its richer connectivity, its performance degrades considerably compared to the Mesh when technological constraints are taken into consideration. Our results also indicate that the 4 Hops variation of the Express Cube is the more suitable for Networks-On-Chip among the Express Cube variations as it is a better fit for designs with large number of cores.

Keywords: Performance evaluation, Network-On-Chip, Mesh, Express Cube, Simulation, Discrete-Event Simulation.

Résumé

L'objectif principal de ce projet est d'évaluer la performance de certaines topologies qui ont été proposées pour les réseaux sur puce, y compris le Mesh et l'Express Cube avec ses variantes. Cette dernière est basée sur un simple réseau d'interconnexion Mesh augmenté par des fils à longs sauts appelés canaux express pour permettre le contournement des sauts intermédiaires pour la communication non locale. Alors que les études existantes se sont concentrées sur les propriétés théoriques des graphes de telles topologies, notre étude examine les performances 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. Des expériences de simulation approfondies ont été réalisées puis analysées à l'aide de méthodes statistiques. Les résultats obtenus révèlent que si l'Express Cube a des performances supérieures lorsque les contraintes technologiques sont ignorées en raison de sa riche connectivité, ses performances se dégradent considérablement par rapport au Mesh une fois les contraintes technologiques sont prises en compte. Nos résultats indiquent également que la variante à 4 sauts du L’Express Cube est la plus adaptée aux réseaux sur puce parmi les variantes du L’Express Cube, car elle convient mieux aux conceptions avec un grand nombre de cœurs.

Mots-clés : évaluation des performances, réseau sur puce, maillage, Express cube, simulation, simulation d'événements discrets.

ملخص

الهدف الرئيسي من هذا المشروع هو تقييم أداء بعض أهم الطبولوجيات المعروفة والتي تم اقتراحها من أجل الشبكات على الرقاقة بما في ذلك Mesh و Express Cube بأشكالها المختلفة. يعتمد هذا الأخير في تركيبته على Mesh معززة بأسلاك طويلة تسمى القنوات السريعة لتمكين تجاوز القفزات الوسيطة للاتصالات الغير المحلية. في حين ركزت الدراسات الحالية على المزايا النظرية للرسم البياني لمثل هذه الطبولوجيات، تفحص دراستنا أداء الشبكات على الرقاقة مع مراعاة القيود التي تفرضها تقنية وتكنولوجيا التنفيذ. أكثر القيود ذات الصلة بالشبكات على الرقاقة هي كثافة الأسلاك الخاصة بها. لتحقيق هدفنا، قمنا بتطوير نموذج محاكاة باستخدام تقنية محاكاة الحدث المنفصل. تم إجراء تجارب محاكاة واسعة النطاق ثم تحليلها باستخدام الأساليب الإحصائية. وتكشف النتائج التي تم الحصول عليها أنه في حين أن Express Cube يتمتع بأداء فائق عندما يتم تجاهل القيود التكنولوجية بسبب اتصاله الأكثر ثراءً، إلا أن أدائه يتدهور بشكل كبير مقارنةً بـ Mesh لما نأخذ القيود التكنولوجية بعين الاعتبار قيد الدراسة. تشير نتائجنا أيضًا إلى أن متغير Hops 4 في Express Cube هو الأكثر ملاءمة للشبكات على الرقاقة من بين متغيرات Express Cube لأنه مناسب بشكل أفضل للتصاميم التي تحتوي على عدد كبير من المعالجات.

الكلمات الرئيسية : تقييم الأداء ، الشبكات على الرقاقة ، Mesh، Express Cube ، المحاكاة ، محاكاة الأحداث المنفصلة.

Contents

List of abbreviations

NoC	Network-on-Chip
SoC	System-on-Chip
MPSoC	Multi-Processor System-on-Chip
DSM	Deep submicron
FT	Fat-Tree
BFT	Butterfly-Fat-Tree
PE	Processing Element
R	Router
DMNI	Direct Memory Network Interface
DMA	Direct Memory Access
NI	Network Interface
X+ X-	Outputs in the X Axis
Y+ Y-	Outputs in the Y Axis
E+ E-	Outputs via Express Channels
λ	Injection Rate (Lambda)
H	Number of Hops
HBE	Number of Hops between the edges
TR	Transmission Time
L	Message Length
RT	Mean Response Time
WT	Mean Waiting Time
TH	Mean Throughput
OMRT	Overall Mean of the Response Time

OMWT	Overall Mean of the Waiting Time
OMTH	Overall Mean of Throughput
SDRT	Standard Deviation of the Response Time
SDWT	Standard Deviation of the Waiting Time
SDTH	Standard Deviation of Throughput
MERT	Margin of Error of the Response Time
MEWT	Margin of Error of the Waiting Time
METH	Margin of Error of Throughput
IR	Injection Rate (L/l)
CI	Constrained implementation
UCI	Unconstrained implementation
EC2H	Express Cube with 2 Hops
EC4H	Express Cube with 4 Hops
EC8H	Express Cube with 8 Hops
ECMH	Express Cube with M Hops
m	Number of Batches used
α	Significance level
$Bmesh$	Bisection width of the Mesh
$Wmesh$	channel width of the Mesh
B_{ECMH}	Bisection width of the ECMH
W_{ECMH}	channel width of the ECMH

List of figures

List of tables

List of Algorithms

General Introduction

We currently live in the era of advanced technology where smart solutions and applications are being deployed widely in different fields every day. This expansion is the result of the evolutions and the improvements of integrated chips which are now used in virtually all electronic equipment and have revolutionized the world of electronics, from computers, servers to smartphones and even Internet of Things (IoT) devices.

Gordon Moore predicted in 1965 the exponential growth of silicon integration and its consequences on the application of integrated circuits. Following this growth, known as Moore's Law, the number of transistors integrated on a single silicon chip has doubled every 18 months, leading to a constant growth in the semi-conductor industry for over 30 years. This technological evolution implied constant changes in the design of digital circuits with, for instance, the advent of gate level simulation and logic synthesis. Amongst these changes, the advent of System on Chip (SoC) represented a major technological shift [1].

A SoC is an integrated circuit composed of many components including: one or several processors, on-chip memories, hardware accelerators, devices drivers, digital/analog converters, and analog components. It was initially named SoC because all the features of a complete "system" were integrated together on the same chip. At that time, a system was dedicated to a single application: video processing or wireless communication for instance. Thanks to the increasing role of software, SoCs are no longer specific, in fact many of them are reused in several different telephony or multimedia devices [1].

The SoC-based system design methodology focuses on the computational aspects of the problem. However, the number of components in a single chip and their performances continue to increase. To address complex real-life applications, it is often required to have multiple processors which can cohesively communicate and provide high parallelism. This, in turn, has resulted in Chip Multi-Processing (CMP) systems to provide scalable computational power. Hundreds of processing cores are integrated on the SoC platform to build Multi-Processor System-on-Chip (MPSoC) in deep submicron (DSM) technology.

In MPSoC systems, the design of communication architecture plays a major role in defining the area, performance and energy consumption of the overall system. In the many-core regime, individual processor speed has improved significantly over the technology generations. As a result, communication architecture has become the roadblock, limiting the overall system performance. Several research groups from academia and industry have started to find out suitable communication architectures for next generation many-core based SoCs. In the process, Network-on-Chip (NoC) has evolved as a standard to design the advanced Multi-Processor Systems-on-Chip (MPSoCs). It provides better predictability, lower power consumption, greater scalability and fault-tolerance compared to the previously known solutions for on-chip communication [2].

The purpose of present project is to revisit the relative performance merit of well-known topologies proposed for NoCs including the mesh and its variation when implementation constraints are taken into account. In order to achieve this, a software simulator has been developed for the most used topologies 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.

Organization of the report

The remainder of the report is organized into four chapters:

- Chapter 1: provides a technical background on NoCs and the important factors that affect their performance, including topology, switching and routing. It also provides an overview on the existing related research work which has compared the performance of various NoC topologies along with a critical summary of these studies

- Chapter 2: explains the simulation model and justifies the selected method of study.

- Chapter 3: presents the system model that has been used for developing the simulator then provides the techniques used for its verification and validation.

- 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.

- Conclusion: summarizes the main conclusions drawn from our comparative study and discusses some possible directions for future research work.

Chapter 1 : Background on networks-on-chip

Some factors determine the performance of the NoC-based system and influence the effectiveness of the related topology implementation. 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

The network-on-Chip topology defines the interconnection pattern among nodes. The usual way of modelling network topologies is as a graph G (N, C) where the vertices, denoted by N represent the set of processing nodes, and the edges C represent the set of links [1]. The links interconnecting the nodes are defined to be bidirectional. Each node is a programmable computer with its own processor, local memory, and other supporting devices. These nodes may have different functional capabilities. For example, the set of nodes may contain vector processors, graphics processors, and I/O processors.

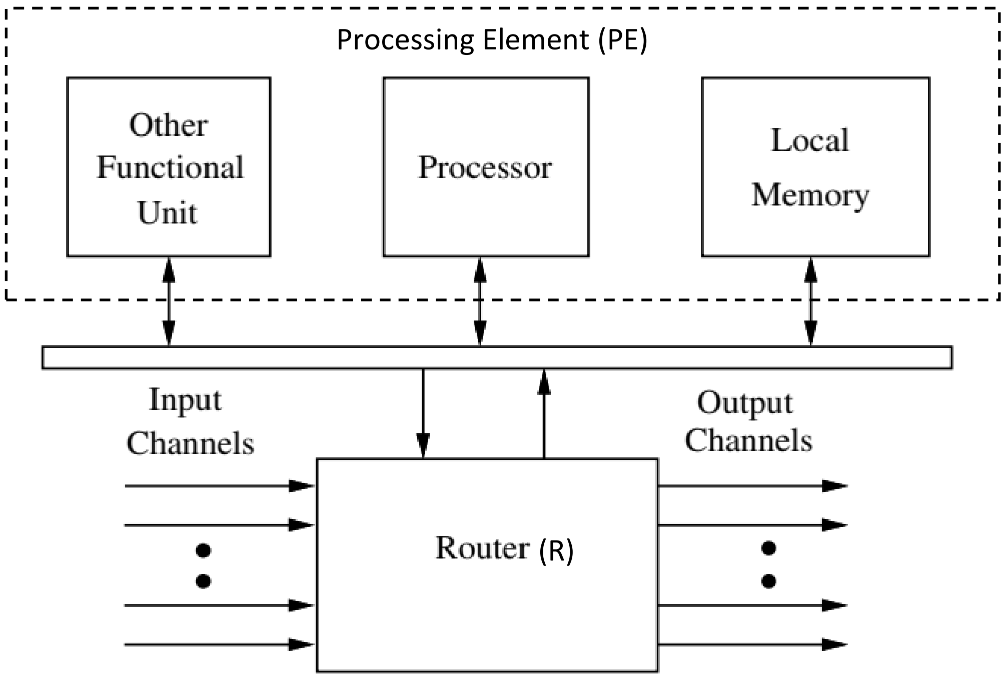


Figure 1-1 A generic node architecture for NoCs

Figure 1-1 shows the architecture of a generic node. A common component of these nodes is a router, which handles message communication among nodes. Each router has direct connections to the router of its neighbours by a bidirectional channel. Although the function of a router can be performed by the local processor, dedicated routers have been used in NoCs, allowing overlapped computation and communication within each node [3]. So, a typical node in the topology is composed of two main elements:

- Processing element (PE): which can be specific-proposed processors, DSPs, memories, or I/O blocks.
- Router (R): which is the responsible of establishing communication and exchanging messages with the other nodes in the NoC.

However, some academic researches proposed new optimized and enhanced designs of the node on the architecture level as shown in what follows:

The proposed Seamless architecture presented in *Figure 1-2 Seamless PE* includes the addition of a hardware Locality Manager to each Processing Element to reduce latency [4].

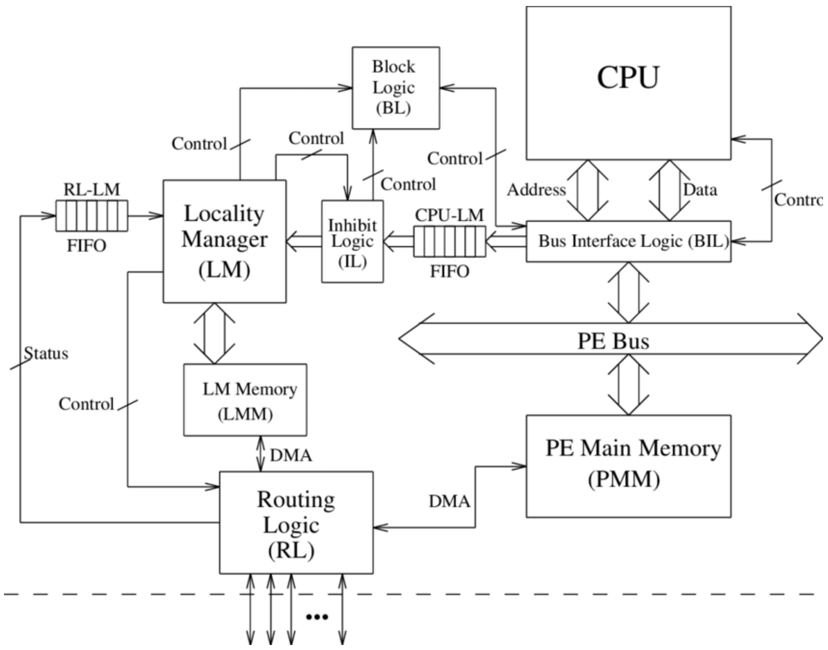


Figure 1-2 Seamless PE [4]

Another specialized design of the NoC-based MPSoC architecture on the communication interface level was proposed to optimize packet reception and transmission, called the DMNI (Direct Memory Network Interface) which merges the functionalities of the DMA and the NI into a single component as shown in *Figure 1-3*. (For more details see [5])

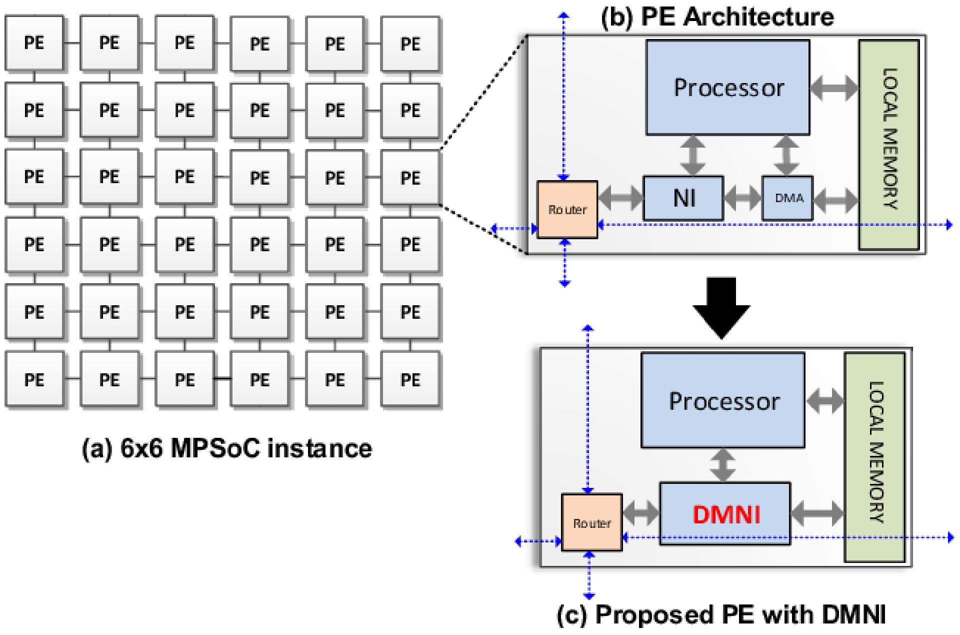


Figure 1-3 DMNI PE Architecture Block [5]

1.1.1 Regular and irregular topologies

In regular topologies, the power consumption and network area scalability with an increase in the size can be predicted. It should be noted that regular network topologies are usually adapted for the majority NoCs. In this section we will focus on the most popular regular topologies along with their advantages and drawbacks mentioned in [6, 7, 8], then we will see the characteristic of irregular ones.

- Ring Topology:

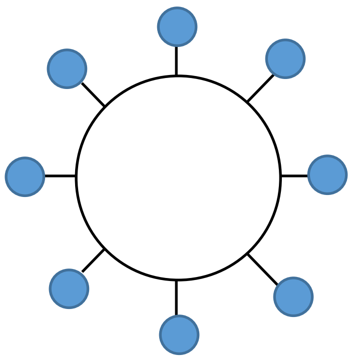


Figure 1-4 Ring Topology

is one of the widely employed NoC topologies. In this topology, a single wire is used to connect each node. Consequently, irrespective of the ring size, each of the nodes has neighbouring nodes as depicted in *Figure 1-4* Based on this, in the ring topology, the degree of each node is two. This implies a corresponding available bandwidth to every node. Although deployment and troubleshooting are comparatively easier, the main drawback of the ring topology is that its diameter increases with an increase in the number of nodes. So, besides the fact that network expansion degrades the performance (scalability issue), ring topology is also prone to a single point of failure (poor path diversity).

- Octagon Topology:

Another prevalent NoC topology is the octagon. A typical octagon topology comprises eight (8) nodes and twelve (12) bidirectional links. Also, just like the ring topology, each node is connected to the preceding and succeeding nodes. So, between a node pair, there are two-hop communications. Also, to route a packet between the network, a simple shortest-path routing can be employed. Besides, compared with a shared bus topology, higher aggregate throughput can be achieved. Furthermore, the architecture can be connected to support bigger designs, resulting in better scalability.

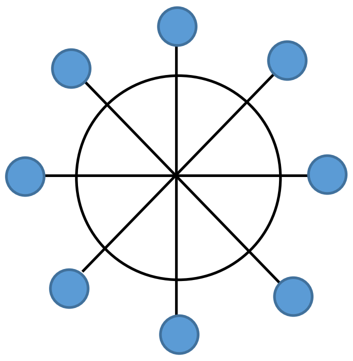


Figure 1-5 Octagon Topology

- Star Topology:

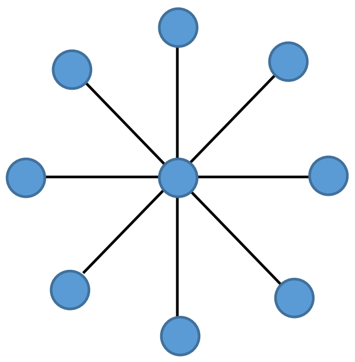


Figure 1-6 Star Topology

The star topology in which the entire nodes are connected to a central node is shown in *Figure 1-6*. Assume an N nodes with N−1 connected nodes to the central node. In this architecture, the central node has an N−1 degree while others have a degree of 1. Therefore, regardless of its size, the star topology diameter is 2. In this regard, its main benefit is the offered simplicity and the presented minimum hop count of two due to the associated small diameter. Although the nodes are separated and free of the potential impact from the failed nodes, the central node failure can result in the entire network failure. Furthermore, as the diameter of the central node increases with the number of nodes, a communication bottleneck can take place in the central node.

- Mesh Topology:

The mesh architecture is the widely employed interconnection topology. A typical 4×4 mesh topology with 16 nodes is illustrated in *Figure 1-7*. Besides the router at the edges, each router in the mesh topology is connected to one computation resource and four neighbouring routers through communication channels. With mesh topology, a huge number of nodes can be incorporated in a regular-shape structure. So, this topology offers an attractive solution for path diversity and scalability. Likewise, this topology can tolerate link failure due to multiple paths that connect a pair of nodes. Nevertheless, one of the main challenges of this topology is that its diameter increases significantly with the number of nodes.

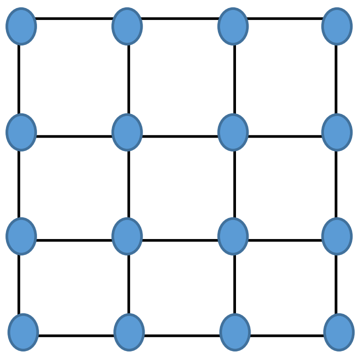


Figure 1-7 Mesh Topology

- Torus Topology:

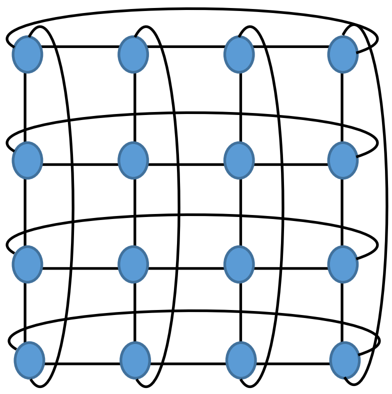


Figure 1-8 Torus Topology

A typical torus topology is depicted in *Figure 1-8*. The architecture is very similar to a mesh topology. However, mesh topology offers a wide diameter. Consequently, the challenge of diameter increase of mesh topology with the network size is addressed by the torus topology. This is achieved through the addition of direct connections between the end nodes that are in the same column or row. For instance, in the torus topology, wrap-around channels are employed for the connection of the edge routers to those at the opposite edge, resulting in a better bisection bandwidth and reduced average number of hops. However, considerable latency is incurred by the torus topology due to the employed lengthy wrap-around connections.

On the other hand, Irregular topologies are based on the integration of various forms, usually regular structures, in different fashions. In this regard, a hybrid, hierarchical, or asymmetric approach can be adopted. Moreover, irregular topologies aim at increasing the available bandwidth compared with the traditional shared busses. Besides, compared with the regular topologies, it helps in reducing the distance among nodes. Also, irregular topologies typically scale nonlinearly with area and power. They are usually based on the concept of clustering and adapted for specific applications. *Figure 1-9* illustrates some irregular topologies such as reduced (optimized) mesh (*Figure 1-9* (i) and (ii)).

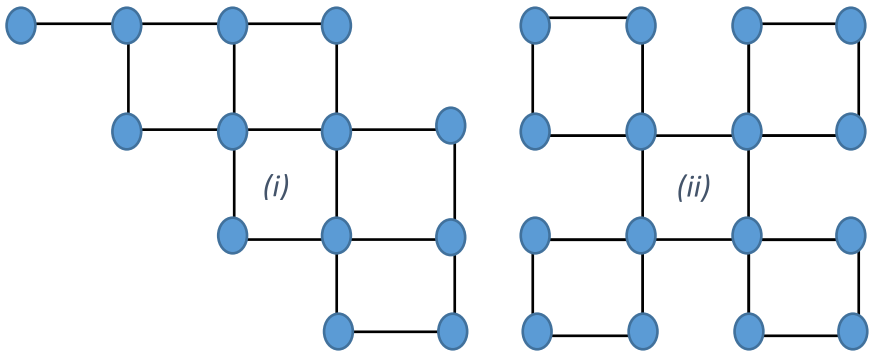


Figure 1-9 Irregular reduced Mesh topologies

1.1.2 The Express Cube Topology

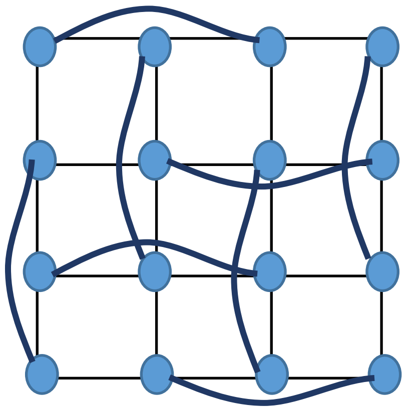


Figure 1-10 Express Cube Topology

An Express Cube network is a Mesh augmented with a number of long (express) channels [9] This would allow non-local messages to traverse these express links and avoid getting delayed at intermediate nodes, and also not add to contention at the ports there. Thus, messages would have dedicated wires for traversing most of the distance, which would reduce latency tremendously and remove the power that these messages would consume at the intermediate nodes.
The trade-off however is that each router would now have multiple ports, which in turn requires a bigger crossbar. Thus, the area of the router goes up along with the power consumption at the crossbar. Moreover, the available metal area adds a limitation on the number of these express cubes that can be added. A 4x4 express cube topology with 2-hop express links is shown in *Figure 1-10* [10]. Express links in the x and y dimensions start at every alternate router. This topology design was chosen to allow many symmetrical express links, without increasing router ports significantly, since the power consumed by the crossbar increases remarkably with the increase in number of ports. This topology has 7 ports per router, as compared to the 5-port routers in the baseline 2D Mesh.

1.2 Topological parameters

In this section, we provide a sight about various factors such as diameter, degree, bisection width and link complexity which are some of the important parameters that characterize and distinguish one topology from the others.

1.2.1 Diameter

The diameter of the network is the maximum distance between any two nodes in the topology, where distance is the number of links in the shortest route [11]. For example, in *Figure 1-4* the Ring topology has a diameter of four, in *Figure 1-7* the Mesh has a diameter of six, in *Figure 1-8* the torus has a diameter of three and for the Express Cube in *Figure 1-10*, the diameter is equal to four. The diameter serves as a proxy for the maximum latency in the topology, in the absence of contention.

1.2.2 Degree

The degree of a topology refers to the number of links at each node. For instance, in *Figure 1-4*, a ring topology has a degree of two since there are two links at each node, while in *Figure 1-8* a Torus has a degree of four as each node has four links connecting it to four neighbouring nodes. Note that in *Figure 1-7* of the mesh network, not all nodes have a uniform degree. The same thing for the Express Cube In *Figure 1-10.*
Degree is useful as a proxy for the network's cost, as a higher degree requires more ports at routers, which increases implementation complexity and adds area/energy overhead at each router [11]. That's the reason why in our case of study, we decide that a node in the Express Cube topology can have a maximum of two express links placed either in the X dimension or in the Y dimension but not both. As a result, the maximum node degree for an Express Cube network is equal to six (four for the base links + two express links).

1.2.3 Cuts and Bisections

A cut of a network, $C(N1, N2)$, is a set of channels that partitions the set of all nodes N^* into two disjoint sets, $N1$ and $N2$. Each element of $C(N1, N2)$ is a channel with a source in $N1$ and destination in $N2$, or vice versa. The number of channels in the cut is $|C(N1, N2)|$ and the total bandwidth of the cut is :

$$B(N1, N2) = \sum_{c \in C(N1,N2)} b_c$$

A bisection of a network is a cut that partitions the entire network nearly in half, such that $|N2| \leq |N1| \leq |N2| + 1$, and also partitions the terminal nodes nearly in half, such that $|N2 \cap N| \leq |N1 \cap N| \leq |N2 \cap N| + 1$. The channel bisection of a network, B_C, is the minimum channel count over all bisections of the network.

$$B_C = \min_{bisections} |C(N1, N2)|$$

The bisection bandwidth of a network, B_B is the minimum bandwidth over all bisections of the network

$$B_B = \min_{bisections} B(N1, N2)$$

We use the bisection bandwidth of a network as an estimate of the amount of global wiring required to implement it [12]. For instance, the *Figure 1-11* presents a cut of 4x4 Mesh while the *Figure 1-13* and *Figure 1-13* present cuts of 8x8 Express Cube with 4 Hops and 2 Hops respectively. We will use these cuts later in chapter 4 to analyse the bisection width of the topologies before performing a simulation under constrained implementation.

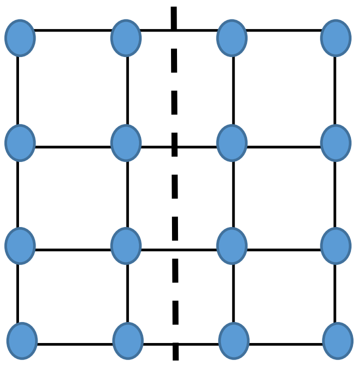


Figure 1-11 Cut of 4x4 Mesh

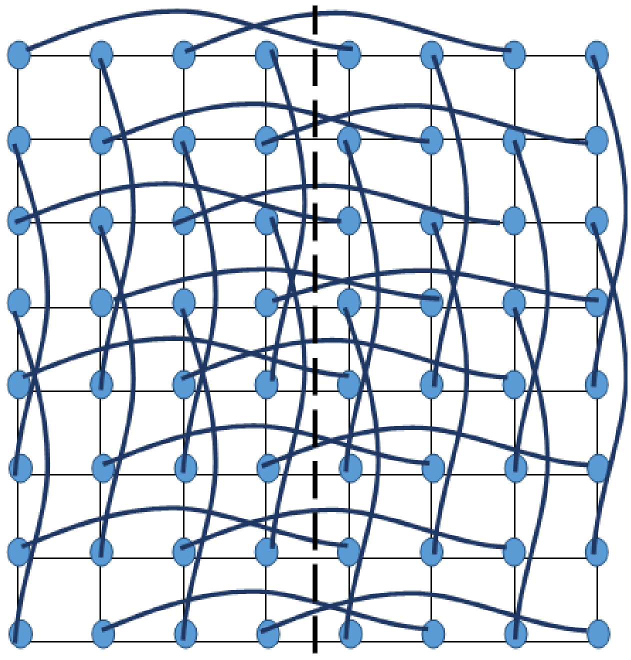


Figure 1-13 Cut of 8x8 Express Cube with 4 Hops

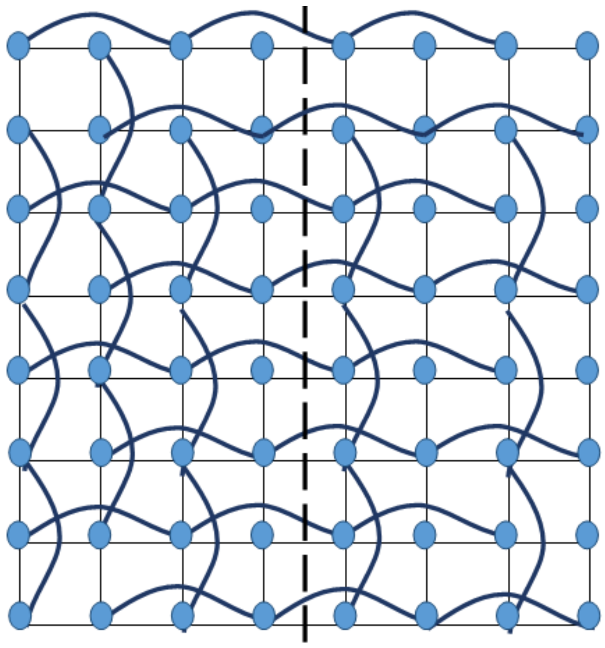


Figure 1-13 Cut of 8x8 Express Cube with 2 Hops

Table 1-1 depicts a comparison of Mesh, Torus and Express Cube based on several graph-theoretic properties with $N = k^n$ nodes where k is the number of nodes per dimensions and n is the number of dimensions.

Network	Degree	Diameter	Average distance	Bisection Width
n-Dimensional Mesh	<=2n	n(k-1)	$\frac{n}{3}(k-\frac{1}{k})$	$2\sqrt{\mathrm{N}}k^{\frac{n}{2}-1}$
n-Dimensional Torus	<=2n	$n\frac{k-1}{2}$	$n(\frac{k-1}{4})$	$\frac{4N}{k}$
n-Dimensional Express cube M Hops	<=2(n+1)	n (Div (k-1, *M*) +Mod (k-1, *M*))	$<\frac{n}{3}(k-\frac{1}{k})$	$2\sqrt{\mathrm{N}}(\frac{M}{2}+1)k^{\frac{n}{2}-1}$

Table 1-1 Comparison of the topological properties of Mesh, Torus and Express Cube

1.3 Related research work

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

1.3.1 2D Mesh vs. 2D Digraph (2012)

The 2D Directed graphs known as Digraphs are a promising alternative for the 2D Mesh topologies [14], this proposal has led the researchers to perform an evaluation of the three well-known digraphs, namely de Bruijn, shuffle-exchange, and Kautz shown in *Figure 1-14* against the most known 2D Mesh topology using an interconnection network simulator that is developed based on the POPNET simulator.

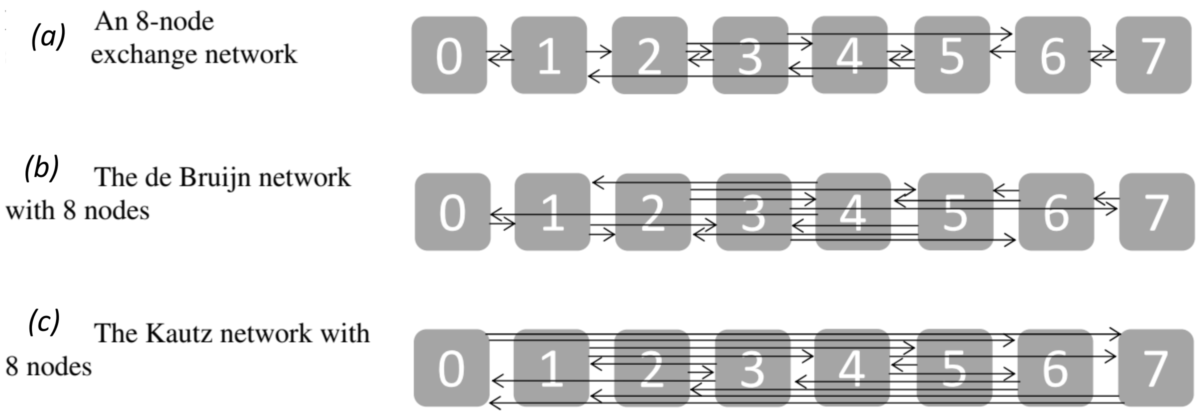

(a)
An 8-node exchange network
(b)
The de Bruijn network with 8 nodes
(c)
The Kautz network with 8 nodes
0
1
2
3
4
5
6
7

Figure 1-14 Digraphs Topologies [14]

The shuffle-exchange network is a popular interconnection topology for multiprocessors and multicomputers due to its scalability and distributed self-routing capability, two nodes are directly connected (via *exchange* bidirectional link which is equivalent to two unidirectional links) to each other if their addresses differ in their least significant bits.

A node is connected to another node (by *shuffle* unidirectional link) whose address is a one-bit cyclic shift of its address. Movement of data between the adjacent nodes via shuffle and exchange links is respectively called shuffle and exchange operations. Messages generated at source nodes can be communicated between network nodes using shuffle and exchange operations until reaching their destination nodes.

The de Bruijn network and the generalized Kautz digraph are another well-known digraph topology for multiprocessors, they are nearly match the shuffle-exchange network.

The performance and power consumption of the proposed topologies are evaluated and compared to those of the equivalent Meshes. The simulation results were obtained for 8 × 8 and 16 × 16 Mesh NoCs, and 8×8 and 16×16 2-D SEM and 2-D BM NoCs.

To simulate the proposed NoC topologies, an interconnection network was developed and designed based on POPNET Simulator. It was modified to mimic the exact operation of the 2D mesh and the 2D digraph-based Mesh NoCs. The simulator was also customized to support the corresponding routing algorithms for the considered networks.

The results obtained exhibit that under uniform traffic, the 2D BM network has the smallest average message latency for both message sizes, because the averages inter-node distance is logarithmic and lower than that of other equivalent networks.

The researchers had noted also that increasing the network size causes earlier saturation in a simple 2-D mesh NoC, the 2-D BM and 2-D SEM networks attain a reduction in message latency with respect to the 2-D mesh network for the full range of network loads.

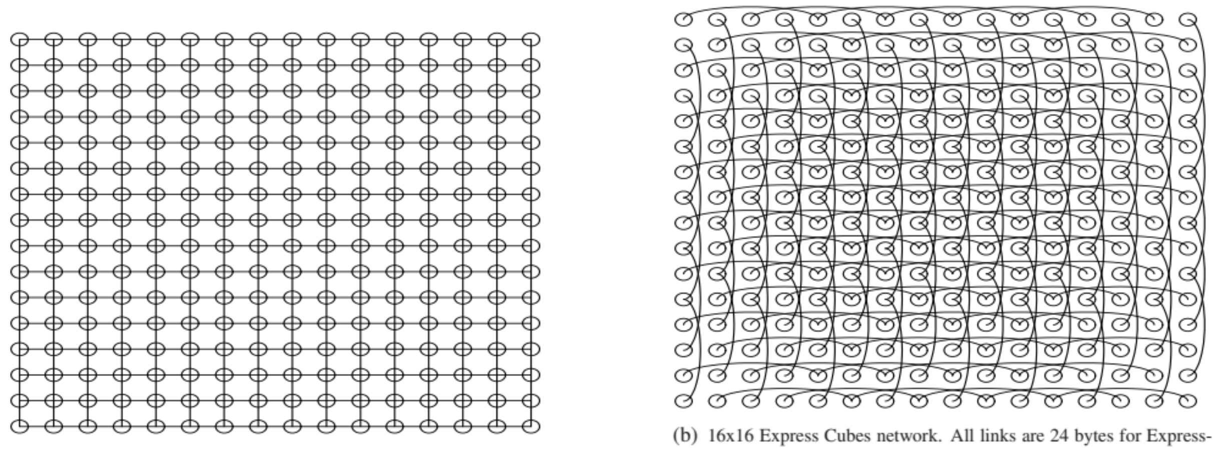
As conclusion the 2-D digraph-based NoCs have superior performance over the 2-D mesh NoC, since the average hop counts traversed by messages in a 2-D digraph-based network is smaller than that in the equivalent 2-D mesh.

The node degrees of the 2-D digraph-based networks and 2-D mesh networks are the same. However, unlike the 2-D mesh topology, the 2-D digraph-based networks have some links that connect non-adjacent nodes and therefore, they may be longer than links in their 2-D mesh counterparts. This can lead to an increase in the network area and also cause difficulties in link placement. The latter can be solved by a number of efficient VLSI layouts proposed for digraphs.

1.3.2 2D Mesh vs. Express Cube (2014)

In this work [10], a comparison study has been carried out between the physical and virtual Express topologies for NoCs, the physical express topologies are the ones that employ long physical links between non-local routers to reduce the effective network diameter in the 2D Mesh baseline, using GARNET software simulation.

The study evaluated a baseline network (2D Mesh) against the express cube topology, the size of networks was chosen to be 16x16 nodes and the express links in the Express cube were 4 hops long like shown in *Figure 1-15* 2D Mesh Baseline vs Express Cube 4 Hops.



(a) 16x16 Baseline and EVC-network. All links are 72 bytes.

(b) 16x16 Express Cubes network. All links are 24 bytes for Express-normal topology and 12 bytes for Express-CDLSI topology. Note that the normal 1-hop links are not shown.

Figure 1-15 2D Mesh Baseline vs Express Cube 4 Hops [10]

The metrics examined in this evaluation were latency, throughput, area and power consumption. The evaluation was performed using Dimension ordered X-Y routing whilst the synthetic traffic was modelled using Uniform Random, Bit Complement and Tornado distributions.

The performance results reported in this study indicate that the Express Cube network have lower latency at low injection rates comparing to the 2D Mesh Baseline. This is because the express network utilizes express long-hop links to "completely" bypass intermediate routers and links and thus have a lower average network latency at low injection rates. While for the throughput, the results indicate also that the express network has a higher saturation throughput than the 2D Mesh Baseline. This is because the bypassing of intermediate routers in the express network leads to lower contention and hence pushes the saturation throughput. The conclusion was that the physical express topologies are desirable for systems that want lowest network latency for low injection rates.

1.3.3 2D Mesh vs. Ruche Network vs. Express Cube (2020)

Ruche Networks have been proposed by Bespoke Silicon Group from University of Washington in 2020 at the 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS) conference [13], Ruche Networks are based on simple 2-D mesh networks but amplify the NoC bandwidth and reduce NoC diameter of tiled architectures by adding long-range physical channels from each tile to other tiles on the same row or column like shown in *Figure 1-16*.

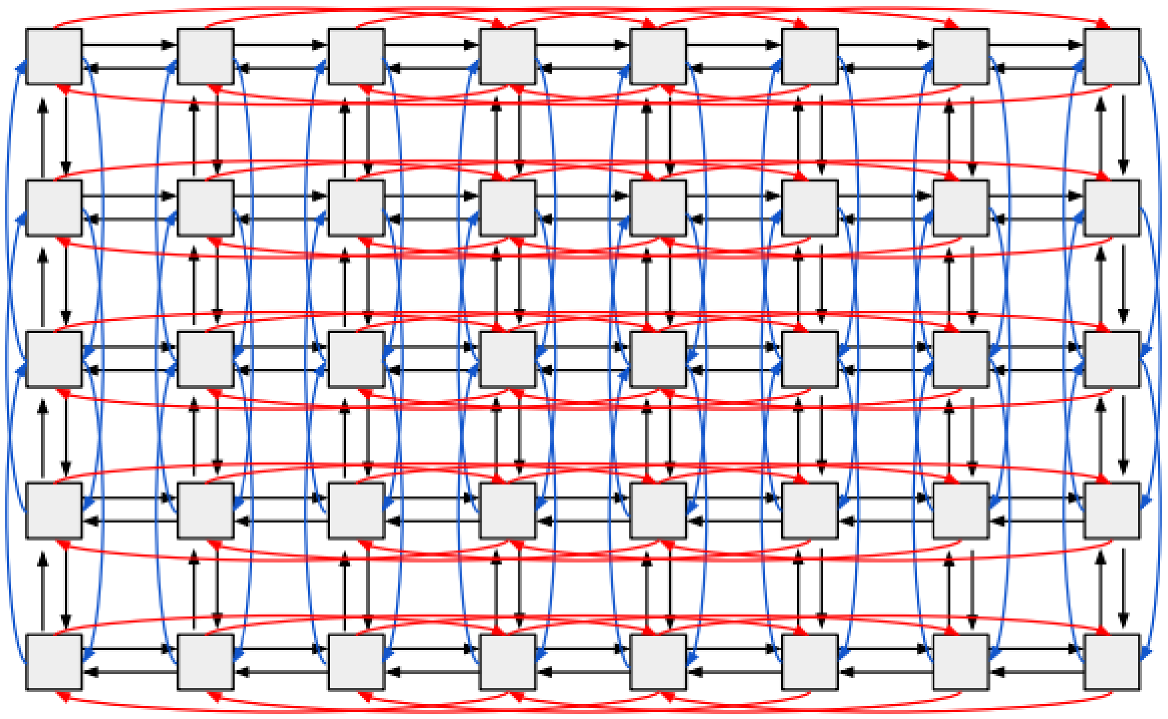


Figure 1-16 A Ruche Network augments a 2D Mesh with additional parallel high-bandwidth links called Ruche Channels [13]

A Half Ruche Network only has Ruche Channels along one dimension (i.e., either X or Y) and a Full Ruche Network has them in both dimensions.

To reduce the router area impact even more, the Depopulated Crossbar Ruche Network, employs a modified router design that eliminates the ability to turn out of a Ruche Channel; e.g., packets must hop off of the Ruche Channel on to the local network before changing dimensions (i.e. to Y or P port). Then the N, S and P crossbars are depopulated to prevent the area overhead of adding the two Ruche ports. Intriguingly, depopulated variants can have superior band width characteristics to fully populated ones, because its better load balances traffic.

The term Ruche Factor describes how far the Ruche Channels span. So, for example, a Ruche Network with Ruche Factor zero is just a 2D Mesh with single links to nearest neighbours; with Ruche Factor one has dual links to nearest neighbours; and with Ruche Factor two has one link to neighbours and another link to the neighbours' neighbour. the Ruche factor is referred in the Express Cube Network as the number of hops *H*.

A performance evaluation to the Ruche Networks has been carried out using a cycle accurate model to calculate the performance characteristics. The results reveal that a Fully Populated Crossbar Ruche Network has the best unloaded latencies, followed by the area-optimized Ruche Network with depopulated crossbar and finally the Express Cube network.

The analysis shows also that the optimal Ruche Factor is also quite small for all networks across all machine size. For a machine size of 16, the depopulated Ruche Network and the Express Cube are both latency optimized with a Ruche Factor of 3 while the fully populated Ruche Network is optimal with a Ruche Factor of 4, outperforming a Ruche Factor of 3 by only 2 cycles. As the machine size increases, so does the optimal Ruche Factor; however, even for a machine of size 128 the depopulated Ruche Network, fully populated Ruche Network and Express Cube are optimal at Ruche Factors of 9, 10, and 7 respectively.

1.3.4 2D Mesh vs. Torus (2020)

The popularity and the simplicity of the 2D Mesh and 2D Torus has made them a common choice for comparisons. In this final year project [15], a comparison study has been carried out between the 2D Mesh and 2D Torus topologies for NoCs by taking into account the constraints imposed by implementation technology using a developed simulation software.

The metrics used to determine the overall performance of the 2D mesh and 2D torus are only the mean throughput and the mean response time. The study used as parameters the message arrival rate, network size and two traffic patterns (Uniform and Hotpot), the routing algorithm used is the deterministic routing -XY- and the implemented switching technique is Packet-Switching.

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 results have revealed also 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.

1.3.5 Summary

Table 1-2 provides a summary of the existing works mentioned in the previous sections.

Authors	Topologies	Metrics	Parameters	Findings
R. Sabbaghi et al. (2012)	**2D Mesh vs. 2D Digraph**	Latency & Power consumption	Modified Version of XY-dimension routing	the 2-D digraph-based NoCs have superior performance over the 2-D mesh NoC
C. O. Chen *et al.* (2014)	**2D Mesh vs Express Cube**	Latency, throughput & Power consumption	Dimension ordered X-Y routing	the Express Cube network have lower latency at low injection rates comparing to the 2D Mesh Baseline
D. C. Jung *et al.* (2020)	**2D Mesh vs Ruche Network vs Express Cube**	Latency	XY-dimension routing & wire length	the depopulated Ruche Network and the Express Cube are both latency optimized
A. Ould-Khaoua and H. Terranti (2020)	**2D Mesh vs Torus**	Mean response time & throughput	packet switching & Deterministic routing	Torus has superior performance due to lower diameter and average distance. However, Mesh is better when constraints are taken into consideration

Table 1-2 Summary of the existing works

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 Express Cube, Mesh versus Ruche Network, Mesh versus Digraph and Mesh versus Torus. However, some of 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.

In addition, some of these comparisons was performed using open-source simulations software, these simulators implement wide variety of functions and modules which made the simulation software somehow difficult to understand and hard to modify it in order to match our needs and our case of study. For that reason, we decide to design and develop a new simulation software from the scratch that fits well to our research study.

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.

Chapter 2 : Simulation modelling

In this chapter we start 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. After that we present the system model that we used in our simulation. Finally, we present the simulation environment used during the development and the execution of the simulation model.

2.1 Justification of the method of study

In order to perform the performance comparison between the competing topologies, firstly we have to select one over of the three main performance evaluation techniques mentioned in [16], which are analytical modelling, simulation and measurement.

The selection choice was made by taking into account some considerations and by analysing important factors like the time required of the study, the cost of the resources needed in the study and the accuracy of the result. In *Table 2-1* we show the defined criteria for selecting an evaluation technique [16].

Criterion	Analytical Modeling	Simulation	Measurement
Stage	Any	Any	Postprototype
Time required	Small	Medium	Varies
Tools	Analysts	Computer Languages	Instrumentation
Accuracy	Low	Moderate	Varies
Trade-off-evaluation	Easy	Moderate	Difficult
Cost	Small	Medium	High
Saleability	Low	Medium	High

Table 2-1 Criteria for selecting an evaluation technique [16]

The 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. Moreover, the studies that analysed the static properties of NoC topologies using for instance graph theory. However, such studies do not consider time dependent behaviour 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.

2.2 Simulation techniques

Among the variety of simulations that are described in the literature, those that would be of interest to computer scientists are Emulation, Monte Carlo Simulation, Trace-Driven Simulation, and Discrete-Event Simulation [16].

2.2.1 Emulation

This simulation technique consists of using hardware or firmware in order to simulates one kind of terminal on another for example. A processor emulator emulates an instruction set of one processor on another. Even though emulation is a type of simulation, the design issues for emulation are mostly hardware design issues. As a result, emulation is not an option for our research study.

2.2.2 Monte-Carlo Simulation

Monte-Carlo simulation is a method of simulating statistical systems. The method uses randomness in a defined system to evolve and approximate quantities without the need to solve the system analytically.

The model predicts by using a range of values in the domain of the problem rather than a specific input. This method leverages distributions of probability (normal, gaussian, uniform, etc.) for any variable which has uncertainty. Based on the number of trials specified, this process of using random values in a domain is repeated numerous times. Generally, the greater number of trials, the higher likelihood the outcome will converge to a value. Commonly used in time series analysis for long term predictive modelling. Once all the simulations are complete, you will have a range of possible outcomes with the associated probability of each result occurring.

In Addition, Monte-Carlo Simulation is static simulation that is used to model probabilistic phenomenon that do not change characteristics over time However, this technique is not suitable for our research study as it cannot model the system behaviour over time.

2.2.3 Trace-Driven Simulation

Trace-Driven simulation consists of two steps. Using a functional simulator or real systems, the program action log is collected and written to a file. This log is called a trace. Depending on what is being collected, and in our case of study the trace may include times of the arriving packets, the states of the buffers and the queues.

The next step is the so-called trace playback, when the simulator reads the trace and executes all the operations and scenarios from there one by one. As a result, it is possible to calculate the response time and get other interesting information like, for example, the throughput and the utilization of channels.

It is worth mentioning that trace execution is deterministic, i.e., the same sequence of actions can be reproduced as many times as needed. By changing the parameters of the model (the size of the nodes, buffers, and queues) and using various internal algorithms or fine-tuning them, one can investigate how a particular parameter affects the overall system performance and which parameter set gives the best results. All of this can be done with a virtual model prototype of the device before creating a hardware prototype.

This approach is rather difficult because it requires application pre-running to collect the trace, and the trace file contains very long sequences which consume a lot of storing and processing capacities. In addition, the ability of changing the trace characteristics to produce and simulate other scenarios is not possible as a result this simulation technique is not the best choice for our case of study.

2.2.4 Discrete-Event Simulation

Modelling complex systems has become a way of life in many fields, most especially in the engineering, health, management, mathematical, military, social, telecommunications, and transportation sciences. It provides a relatively low-cost way of gathering information for decision making. Since the size and complexity of real systems in these areas rarely allow for analytical solutions to provide the information, discrete-event simulation executed on a computer has become the method of choice [17].

This technique compresses time so that years of activity can be simulated in minutes or, in some cases, seconds. This ability enables an investigator to run through several competing operational designs in a very small fraction of the time required to try each on the real system. In addition, the purpose of discrete event simulation is to analyse the behaviour of the system in which state changes over time and can be represented by a collection of discrete events.

Every discrete-event system has a collection of state variables that change values as time elapses. A change in a state variable is called an *event*, and this concept is the basic building block of every discrete-event simulation model.

This simulation technique is the most suitable technique for our research study for its flexibility and simplicity in designing and building the simulation model.

2.3 System model

For the purpose of our study, the NoCs are modelled 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 $S=k^n$. For example, in the case of a 2D topology the address for any given node is designated as *[x][y]* with $0 \le x,y < k$.

2.3.1 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 2-1*.

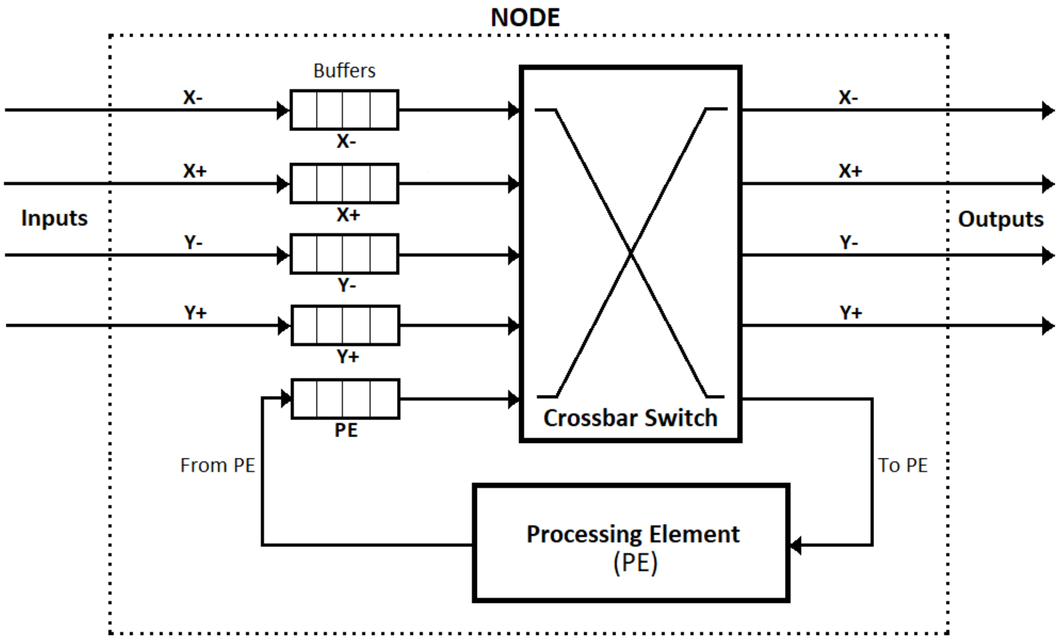


Figure 2-1 A Node structure in the 2D Mesh

In the Express Cube NoC, the express nodes have the same node structure as in the 2D Mesh except that there are two additional input buffer queues (E- and E+) and two additional express output channels, they are highlighted in red in *Figure 2-2*.

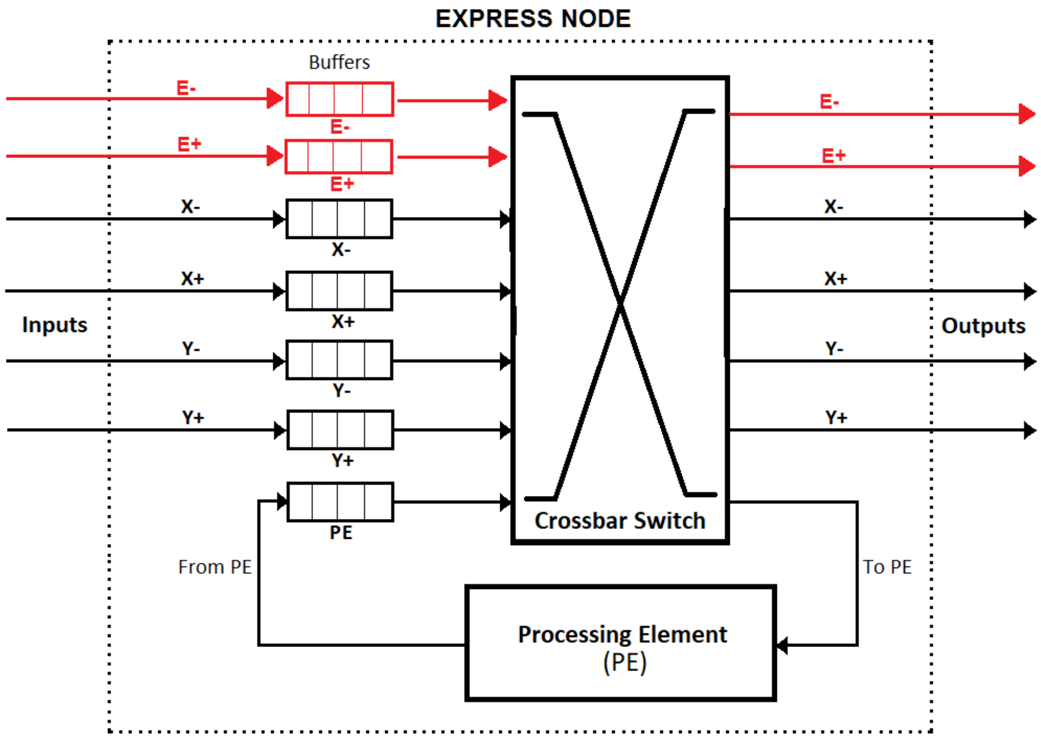


Figure 2-2 An Express Node structure in the Express Cube

2.3.2 Switching and routing

The nodes use packet switching with input buffer queues of large capacity. This is realistic due to Moore's Law, 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 2-3* 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, and more importantly it avoids the issue of deadlock during message routing.

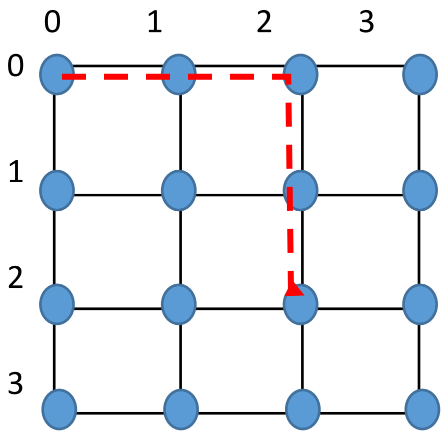
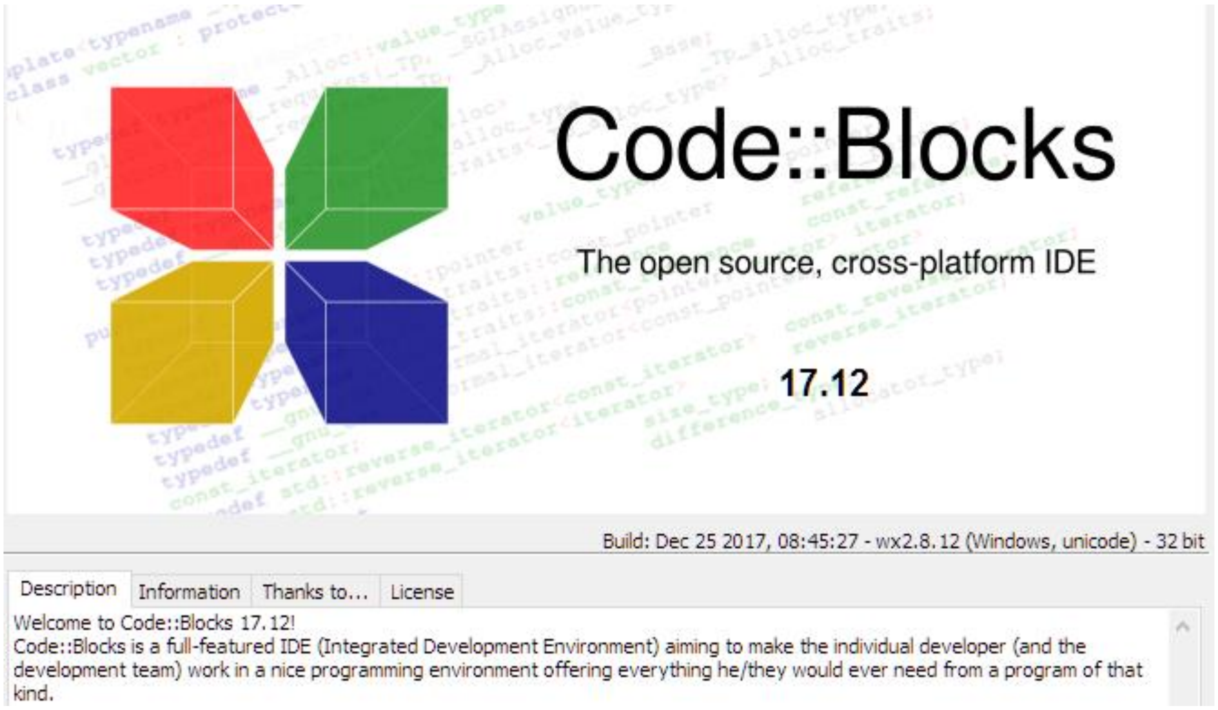


Figure 2-3 An example of a message route from source to destination using deterministic routing in the 2D mesh

2.4 Simulation environment

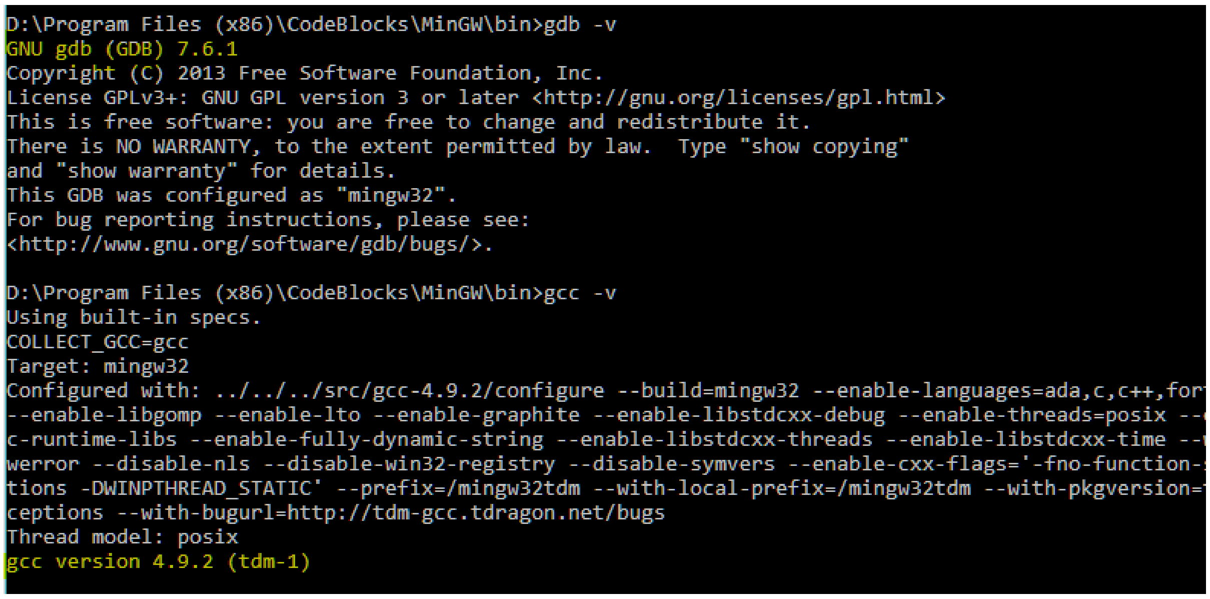
The simulation model is developed then executed using the low-level programming language C in the Codeblocks Integrated Development Environment Installed on a computer running Windows 10 and have 12 GO of RAM. The compiler used is *gcc* and the debugger used to identify and fix bugs is gdb. The *Figure 2-4* and *Figure 2-5* respectively present the versions of the Codeblocks, the debugger and the complier used in the development.



Build: Dec 25 2017, 08:45:27 - wx2.8.12 (Windows, unicode) - 32 bit

Welcome to Code::Blocks 17.12!
Code::Blocks is a full-featured IDE (Integrated Development Environment) aiming to make the individual developer (and the development team) work in a nice programming environment offering everything he/they would ever need from a program of that kind.

Figure 2-4 Snapshot of the version of CodeBlocks used in the development



```
D:\Program Files (x86)\CodeBlocks\MinGW\bin>gdb -v
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.

D:\Program Files (x86)\CodeBlocks\MinGW\bin>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
Target: mingw32
Configured with: ../../../src/gcc-4.9.2/configure --build=mingw32 --enable-languages=ada,c,c++,for
--enable-libgomp --enable-lto --enable-graphite --enable-libstdcxx-debug --enable-threads=posix --
c-runtime-libs --enable-fully-dynamic-string --enable-libstdcxx-threads --enable-libstdcxx-time --
werror --disable-nls --disable-win32-registry --disable-symvers --enable-cxx-flags='-fno-function-
tions -DWINPTHREAD_STATIC' --prefix=/mingw32tdm --with-local-prefix=/mingw32tdm --with-pkgversion=
ceptions --with-bugurl=http://tdm-gcc.tdragon.net/bugs
Thread model: posix
gcc version 4.9.2 (tdm-1)
```

Figure 2-5 Snaphot of the versions of the gdb and gcc used in the development

Chapter 3 : Implementation of the simulation model

In this chapter we discuss some of the important aspects in the implementation of the simulation model that is designed to mimic the exact operations of the 2D mesh and the Express Cube NoCs.

All discrete-event simulations have a common structure. Regardless of the system being modelled, the simulation will have some of the components described in [16].

Our simulator is enhanced by a tracking module that allows us to track the lifecycles of messages across the network during the simulation. Also, it generates a summary for the obtained results in a pre-formatted format to facility the statistical data collection and the analysis of the simulation results.

In this section we present the pseudo code for the main program of the simulator along with a description of the events involved in the simulation. The events and the pseudo code described below applies to the Express Cube with different hops, however the 2D Mesh topology share similar characteristics except with differences in some events. We will present also in the end of the section the techniques used in the verification and the validation of the simulation model.

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, reports generation, dynamic memory management and the models of queues have not been included due to space limitations.

3.1 Data structures

The different data structures implemented in the simulation model are:

- ***Event:*** stores information related to a single event in the simulation. It contains a field for the type of the event, its time, the location (i.se. the node where the event will occur), the input and the output.

- ***EventQueue*:** Is an ordered linked list queue that stores events that occur during the simulation. The events in this queue are sorted in order of time, the first entry in the list is the next earliest event. Thus, removal is straightforward. To insert a new event, the list is searched to find the right place for the new entry.

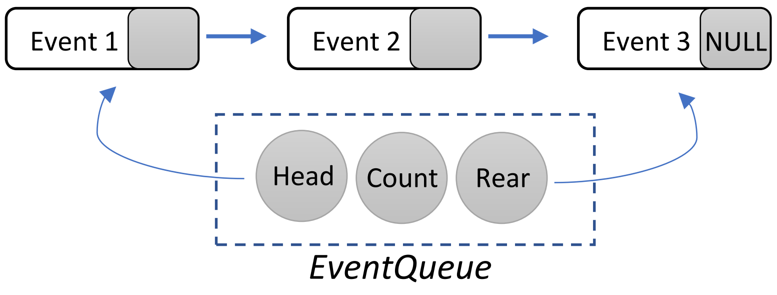


Figure 3-1 Implementation of the EventQueue

- ***Message*:** stores information related to a single message. It contains a field for the ID of the message, a field for its time of arrival, a field for its destination and two fields to track its response time and its waiting time.

- ***MessageQueue*:** is also a linked list queue that stores messages. This queue operates on a FIFO (First In, First Out) principle and does not sort the messages. The insert operation adds a new node after rear and moves rear to the next node. However, the removal operation removes the head node and moves head to the next node.

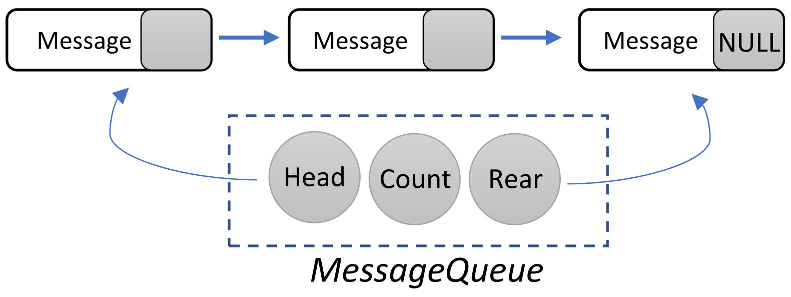


Figure 3-2 Implementation of the MessageQueue

- ***Node*:** represents a single node in the 2D mesh. A node contains 2 MessageQueues per dimension of the topology and an additional queue for the PE, thus totalling 5 queues in the case of the 2D mesh as showed previously in *Figure 2-1*.

- ***NodeEx*:** represents a single node in the Express Cube topology. An express node contains the same MessageQueues as in the 2D Mesh node and 2 additional queues for the express channels, thus totalling 7 queues in the case of the Express Cube. It contains also 4 Express Boolean variables that indicate the existence and the direction of the express links in the node as showed previously in *Figure 2-2*.

The entire topology is represented as matrix of size NxM nodes in the case of 2D Mesh and of size NxM express nodes in the case of the Express Cube.

Both types of nodes implement arrays associated to the outputs to handle requests in the case when the requested output channels are not *IDLEs*. They also implement state variables that reflect the states of the output links.

3.2 Simulation Clock and Time-advancing Mechanism

We use a global variable ***Tnow*** that represent the simulated time, the approach implemented for the advancing of this time is known as the event-driven approach which consist of incrementing the time directly to the time of the next earliest occurring event.

3.3 Simulation events

We have four types of events in the simulator:

- **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

The event **DecideRoute** should have the **EndDecideRoute** event but we assumed in our study that the decision time is equal to *ZERO* and so there is no time between the **DecideRoute** and **EndDecideRoute** in the simulation model.

Each event of the four events mentioned above 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, the mean waiting time and the throughput. In what follows we will describe the procedures mentioned in more detail.

3.3.1 Main Program

```
Main () {
  for (different injection rates λ ) {
      Initialization ();
      while (number of transferred messages < max) {
            Event = Get event from the event queue;
            Tnow = time of Event;
            switch ( type of Event ) {
                  case Arrival:          Arrival (x, y);
                  case DecideRoute:      DecideRoute (x, y, input);
                  case StartTransmit:    StartTransmit (x, y, input, output);
                  case EndTransmit:      EndTransmit (x, y, input, output);
            }
      }
      ReportStatistics ();
  }
}
```

Algorithm 3-1 Main Program

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.

3.3.2 Initialization

```
Procedure Initialization () {

    Tnow = 0;
    Initialize EventQueue;
    Initialize StatisticVars;

    for (i=0; i<N; i++) {
        for (j=0; j<M; j++) {

            Initialize PE queue;
            Initialize X queues;
            Initialize Y queues;
            Initialize E queues;

            Set output States to IDLE;

            Set requests for PE output to IDLE;
            Set requests for X+ output to IDLE;
            Set requests for X- output to IDLE;
            Set requests for Y+ output to IDLE;
            Set requests for Y- output to IDLE;
            Set requests for E+ output to IDLE;
            Set requests for E- output to IDLE;

        }
    }
    PlacingExpressLinks ();

    For Each Node in the Topology Schedule Arrival at t=Tnow
}
```

Algorithm 3-2 Initialization

The Initialization procedure set the initial state of the system state variables and schedule the first primary events of the simulation. The system variables are initialized as follows:

- Tnow is the global clock which is set to 0 (i.e. the start of the simulation time).
- The *EventQueue* initialization includes allocation of memory, setting Head and Rear to NULL and Count to 0.
- The *StatisticVars* initialization includes setting the mean response time, the mean waiting time, the throughput and the number of transferred messages to 0.
- The *InputQueues* initialization includes allocation of memory, setting Head and Rear to NULL and Count to 0.

In the case of the Express Cube, we have the procedure ***PlacingExpressLinks ()*** that is called to set the express links in the topology and updates the Express Boolean variables in the nodes in function of number of Hops ***H*** that the express links has to jump and number of Hops between the edges ***HBE***, these express variables help the router later to make routing decisions by indicating if the current node has an express link in a specific direction or not.

```
Procedure PlacingExpressLinks () {

    for (j=0; j<M; j++) {        ------------------- // Placing Express Links in Lines
      for (i=0; i<N; i++) {

          if ( i is even & j is even & j+H <= HBE ) {
                Node at [i][j].HasExpressX+ = true;
                Node at [i][j+H]. HasExpressX- =true;
          }

          if ( i is odd & j is odd & j+H <= HBE ) {
                Node at [i][j].HasExpressX+ = true;
                Node at [i][j+H]. HasExpressX- =true;
          }

      }

    }


    for (i=0; i<N; i++) {        ------------------- // Placing Express Links in Columns
      for (j=0; j<M; j++) {

          if ( i is odd & j is even & i+H <= HBE ) {
                Node at [i][j].HasExpressY+ = true;
                Node at [i+H][j]. HasExpressY- =true;
          }
          if ( i is even & j is odd & i+H <= HBE ) {
                Node at [i][j].HasExpressY+ = true;
                Node at [i+H][j]. HasExpressY- =true;
          }
      }

    }

}
```

Algorithm 3-3 PlacingExpressLinks

3.3.3 Arrival

```
Procedure Arrival (x, y) {
    Create new message;
    Select random destination for message;
    Place message in PE queue;
    if (message at the head of the queue) {
        Schedule DecideRoute at t=Tnow
    }
    Schedule Arrival at t = Tnow - (lambda*log(1-r));
}
```

Algorithm 3-4 Arrival

The Arrival procedure simulates the generation of new message in the node located at [x][y]. Creating a new message is done by an allocation of memory then an initialization of the variables of the generated message as fellow: time of response and time of waiting to 0, and time of arrival and time of last queuing to *Tnow*.

The destination of the message is randomly selected according to the traffic pattern used (Uniform or Hotspot).

The message is then placed in the PE queue of node [x][y]. If the message is at the head of the PE queue, then a DecideRoute is scheduled at time = *Tnow*.
The scheduling of the following **Arrival** event in the same node will follow a Poisson distribution and it scheduled 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.

3.3.4 DecideRoute

Once the message is at the head of a queue and is ready for transmission, the DecideRoute procedure 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 where the channel is 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.

In the Express Cube topology, there is some additional tests to perform before the making of the routing decision like checking whether the current node has an express links in that direction, and if the difference between the destination and the current node is greater than ***H*** hops. If these conditions are satisfied then the message will be routed through an express channel. Otherwise, the message will be routed through basic channels like in the 2D Mesh.

In the 2D Mesh Topology, DecideRoute is implemented as fellow:

```
Procedure DecideRoute (x, y, input) {
        Message = Get the head of InputQueue;
        if (DestinationX of Message = x & DestinationY of Message = y) {
                if (State of PE= BUSY)
                                Set a request for PE output;
                else
                Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=PE;
        }
        else if (DestinationX of Message > x) {
                if (State of X+ = BUSY)
                                Set a request for X+ output;
                else
                Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=X+;
        }
        else if (DestinationX of Message < x) {
                if (State of X-= BUSY)
                                Set a request for X- output;
                else
                Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=X-;
        }
        else If (DestinationY of message > y) {
                if (State of Y+ = BUSY)
                                Set a request for Y+ output;
                else
                Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=Y+;
        }
        else If (DestinationY of message < y) {
                if (State of Y- = BUSY)
                                Set a request for Y- output;
                else
                Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=Y-;
        }
}
```

Algorithm 3-5 DecideRoute

In the Express Cube of ***H*** Hops Topology, DecideRoute will be implemented as fellow:

```
Procedure DecideRoute (x, y, input) {
    Message = Get the head of InputQueue;
    if (DestinationX of Message = x & DestinationY of Message = y) {
        if (State of PE= BUSY)
            Set a request for PE output;
        else
        Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=PE;
    }
    else if (DestinationX of Message > x) {
        if (DestinationX of Message – x >= H & Current Node HasExpressX+) {
            if (State of E+ = BUSY)
                Set a request for E+ output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=E+;
        } else {
            if (State of X+ = BUSY)
                Set a request for X+ output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=X+;
        }
    }
    else if (DestinationX of Message < x) {
        if (DestinationX of Message – x <= H & Current Node HasExpressX-) {
            if (State of E-= BUSY)
                Set a request for E- output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=E-;
        } else {
            if (State of X-= BUSY)
                Set a request for X- output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=X-;
        }
    }
    else If (DestinationY of message > y) {

        if (DestinationY of Message – y >= H & Current Node HasExpressY+) {
            if (State of E+ = BUSY)
                Set a request for E+ output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=E+;
        } else {
            if (State of Y+ = BUSY)
                Set a request for Y+ output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=Y+;
        }
    }

    else If (DestinationY of message < y) {

        if (DestinationY of Message – y <= H & Current Node HasExpressY-) {
            if (State of E- = BUSY)
                Set a request for E- output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=E-;
        } else {
            if (State of Y- = BUSY)
                Set a request for Y- output;
            else
            Schedule StartTransmit at t = Tnow with x=x, y=y, input=input and output=Y-;
        }
    }
}
```

Algorithm 3-6 DecideRouteEx

3.3.5 StartTransmit

```
Procedure StartTransmit (x, y, input, output) {
    switch ( output ) {
        case PE:    Set PEState to BUSY;
        case X+:    Set X+State to BUSY;
        case X-:    Set X- State to BUSY;
        case Y+:    Set Y+State to BUSY;
        case Y-:    Set Y- State to BUSY;
        case E+:    Set E+State to BUSY;
        case E-:    Set E- State to BUSY;
    }
    Schedule EndTransmit at t = Tnow + TR with x=x, y=y, input=input and output=output

}
```

Algorithm 3-7 StartTransmit

The **StartTransmit** procedure prepares the message to be transmitted via the pre-selected output from the **DecideRoute** procedure, the preparation includes:

- Setting up the chosen output to BUSY all along the transmission phase.
- Scheduling an **EndTransmit** with time= *Tnow* + transmission time of the message ***TR***.
- if the output is **PE** so **EndTransmit** is scheduled with time = *Tnow* only *(TR=0).*

3.3.6 EndTransmit

Procedure EndTransmit (x, y, input, output) {

```
switch ( output ) {
    case PE:
        Set PEState to IDLE;
        switch (input = {X+, X-, Y+, Y-, E+, E-}) {
            Remove the message from the corresponding input queue of node[x][y];
            Collect the response time and the waiting time of the message;
        }
        Check requests for PE for node[x][y];
        if (there is request for output PE from node[x][y] from E+, E-, X+, X-, Y+ or Y-)
            Schedule StartTransmit at t= Tnow, input= the requested input queue, output= PE;
    Break;

    case X+:
        Set X+State to IDLE;
        switch (input = {PE, X+, E+}) {
            Remove the message from the corresponding input queue of node[x][y];
            Place message in X+ queue of node[x+1][y];
            if (message is at the head of X+ queue in node [x+1] [y])
                Schedule DecideRoute at t= Tnow and input= X+ in node [x+1][y];
        }
        Check requests for X+ for node[x][y];
        if (there is request for output X+ from node[x][y] from PE, E+, X+)
            Schedule StartTransmit at t= Tnow, input= the requested input queue, output= X+;
    Break;

    case X-:
        Set X-State to IDLE;
        switch (input = {PE, X-, E-}) {
            Remove the message from the corresponding input queue of node[x][y];
            Place message in X- queue of node[x-1][y];
            if (message is at the head of X- queue in node [x-1] [y])
                Schedule DecideRoute at t= Tnow and input= X- in node [x-1][y];
        }
        Check requests for X- for node[x][y];
        if (there is request for output X- from node[x][y] from PE, E-, X-)
            Schedule StartTransmit at t= Tnow, input= the requested input queue, output= X-;
    Break;
```

```
case Y+:
    Set Y+State to IDLE;
    switch (input = {PE, X+, X-, Y+, E+, E-}) {
            Remove the message from the corresponding input queue of node[x][y];
            Place message in Y+ queue of node[x][y+1];
            if (message is at the head of Y+ queue in node [x][y+1])
                Schedule DecideRoute at t= Tnow and input= Y+ in node [x][y+1];
    }
    Check requests for Y+ for node[x][y];
    if (there is request for output Y+ from node[x][y] from PE, E+, E-, X+, X-, Y+)
        Schedule StartTransmit at t= Tnow, input= the requested input queue, output= Y+;
Break;

case Y-:
    Set Y-State to IDLE;
    switch (input = {PE, X+, X-, Y-, E+, E-}) {
            Remove the message from the corresponding input queue of node[x][y];
            Place message in Y- queue of node[x][y-1];
            if (message is at the head of Y- queue in node [x][y-1])
                Schedule DecideRoute at t= Tnow and input= Y- in node [x][y-1];
    }
    Check requests for Y- for node[x][y];
    if (there is request for output Y- from node[x][y] from PE, E+, E-, X+, X-, Y-)
        Schedule StartTransmit at t= Tnow, input= the requested input queue, output= Y-;
Break;

case E+:
    Set E+State to IDLE;
    switch (input = {PE, X+, X-, Y+, E+,}) {
            Remove the message from the corresponding input queue of node[x][y];
            if (node [x][y] Has Express Link in the X Dimension)
                    Place message in E+ queue of node[x+H][y];
            else    Place message in E+ queue of node[x][y+H];
            if (message is at the head of E+ queue in node[x+H][y] or node[x][y+H])
                Schedule DecideRoute at t= Tnow and input= E+ in the next node;
    }
    Check requests for E+ for node[x][y];
    if (there is request for output E+ from node[x][y] from PE, E+, X+, X-, Y+)
        Schedule StartTransmit at t= Tnow, input= the requested input queue, output= E+;
Break;
```

```
    case E-:
       Set E-State to IDLE;
       switch (input = {PE, X+, X-, Y-, E-,}) {
              Remove the message from the corresponding input queue of node[x][y];
              if (node [x][y] Has Express Link in the X Dimension)
                        Place message in E- queue of node[x-H][y];
              else    Place message in E- queue of node[x][y-H];
              if (message is at the head of E- queue in node[x-H][y] or node[x][y-H])
                Schedule DecideRoute at t= Tnow and input= E- in the next node;
       }
       Check requests for E- for node[x][y];
       if (there is request for output E+ from node[x][y] from PE, E-, X+, X-, Y-)
         Schedule StartTransmit at t= Tnow, input= the requested input queue, output= E-;
    Break;
 Break;
}
```

Algorithm 3-8 EndTransmit

The **EndTransmit** procedure consist of ending the transmission of a message by moving the later 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, then it's sent to the local PE where the statistics are collected.

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*.

In the case of Express Cube where a node has 7 inputs, each input is connected to 6 outputs plus the local PE output making a totalling of 7 outputs, so the number of cases we have to handle is 7x7 = 49 cases, however the **EndTransmit** handles only 34 cases, the 15 remain cases cannot be happened because of the constrained imposed by the routing algorithm.

For example, when the input of a message is Y- or Y+ which means that the message is being transmitted in the meanwhile via Y Axis, so we won't across outputs for X Axis like X- or X+, that because the implemented routing protocol impose that the message has to be transmitted firstly via the X Axis then via the Y Axis. Another common example where a message generated in the PE input cannot be transmitted to the PE output in the same node.

3.4 Model verification

Verification is the process of determining that a model or simulation implementation and its associated data accurately represent the developer's conceptual description and specifications [18]. And in order to verify the correctness of the simulation model we used one of the verification techniques mentioned in [16] and which is the Top-Down Modular design.

3.4.1 Top-Down Modular Design:

This technique requires that the simulation model be divided and structured into modules that communicate to each other via well-defined interfaces.
The interface consists of a number of input and output variables or data structures.

In our case, the simulation model was structured as follows :

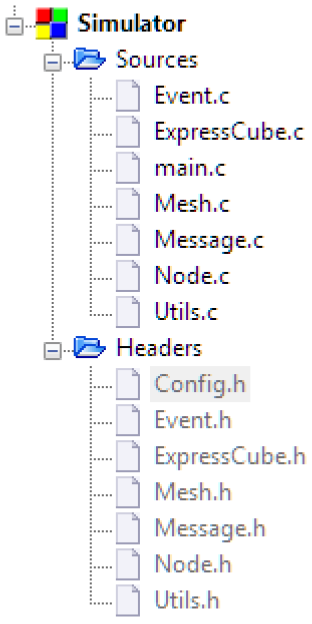


Figure 3-3 Simulator Project Structure

Each one of these modules is composed of two files:

- Source file (****.c***) that implement the procedures and the subprograms of the module.
- Header file (****.h***) containing declarations, macro and data structures definitions that are specified to the module.

Using this approach in designing the simulation model gave us more flexibility and smoothness while developing, debugging and maintaining the modules independently.
That allows the verification of the simulation to be broken down into smaller problems of verifying the modules and their interfaces.

3.5 Model validation

Validation is the process of determining the degree to which a simulation model and its associated data are an accurate representation of the real world from the perspective of the intended uses of the model [18]. And in order to validate the simulation model, we have tested it and run it through some predictable and simplified scenarios in the beginning then through more complex cases to analyse its behaviour and its closeness to real systems.

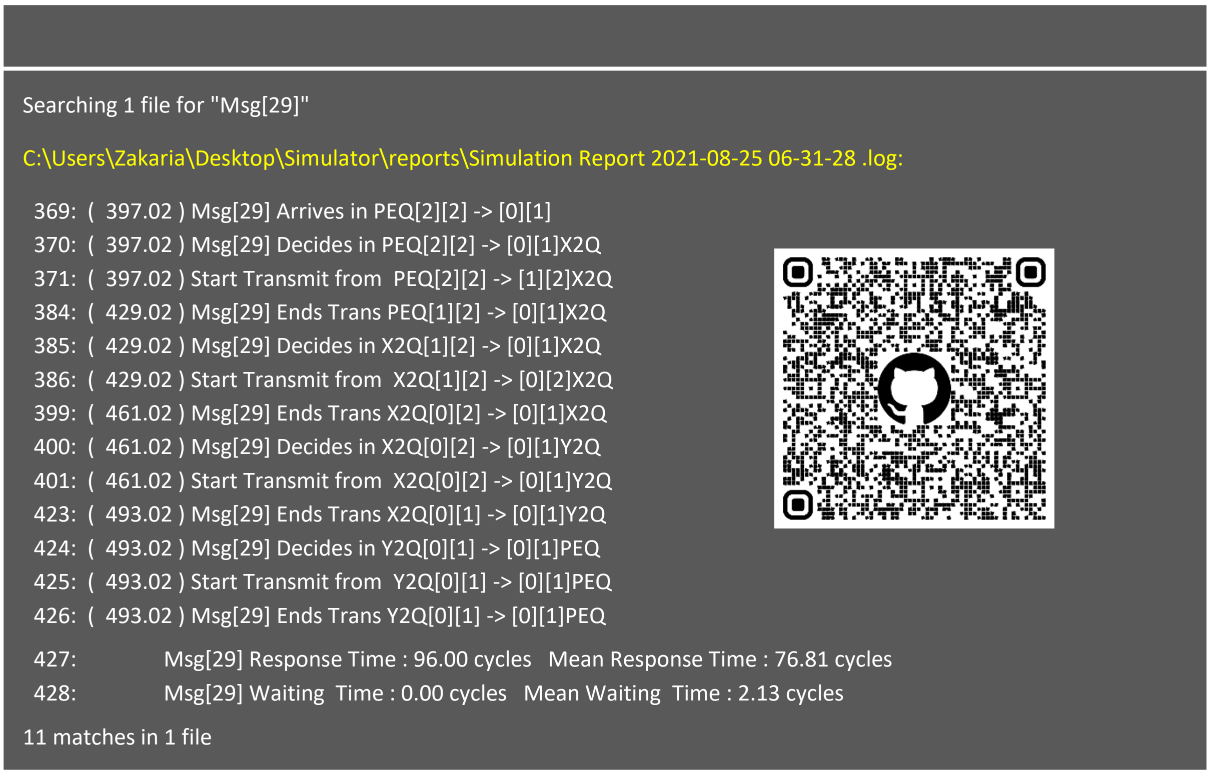
3.5.1 Run Simplified Cases

We run a simplified simulation for a small number of messages (Max = 100) with the ***Logging*** setting to true in the configuration file in order to allow the report generator module to track the lifecycles of all messages crossing the network.

The simulation is performed under the following parameters:

- Topology: 4 x 4 Mesh
- Routing: Dimension-ordered (DOR)
- Traffic: Uniform Random
- Message length: 32 phits
- Injection Rate: 300
- Transmission Time: 32 Cycles

We open the generated log file in *"reports/"* folder, this file contains all the lifecycles of the messages, we filter the result to show the lifecycle related to one specific message, in this case we select the message with ID = 29, Noted as *Msg[29]*.
Note that you can select any other message and that you can check the file for more details.



```
Searching 1 file for "Msg[29]"

C:\Users\Zakaria\Desktop\Simulator\reports\Simulation Report 2021-08-25 06-31-28 .log:

 369:  ( 397.02 ) Msg[29] Arrives in PEQ[2][2] -> [0][1]
 370:  ( 397.02 ) Msg[29] Decides in PEQ[2][2] -> [0][1]X2Q
 371:  ( 397.02 ) Start Transmit from  PEQ[2][2] -> [1][2]X2Q
 384:  ( 429.02 ) Msg[29] Ends Trans PEQ[1][2] -> [0][1]X2Q
 385:  ( 429.02 ) Msg[29] Decides in X2Q[1][2] -> [0][1]X2Q
 386:  ( 429.02 ) Start Transmit from  X2Q[1][2] -> [0][2]X2Q
 399:  ( 461.02 ) Msg[29] Ends Trans X2Q[0][2] -> [0][1]X2Q
 400:  ( 461.02 ) Msg[29] Decides in X2Q[0][2] -> [0][1]Y2Q
 401:  ( 461.02 ) Start Transmit from  X2Q[0][2] -> [0][1]Y2Q
 423:  ( 493.02 ) Msg[29] Ends Trans X2Q[0][1] -> [0][1]Y2Q
 424:  ( 493.02 ) Msg[29] Decides in Y2Q[0][1] -> [0][1]PEQ
 425:  ( 493.02 ) Start Transmit from  Y2Q[0][1] -> [0][1]PEQ
 426:  ( 493.02 ) Msg[29] Ends Trans Y2Q[0][1] -> [0][1]PEQ

 427:            Msg[29] Response Time : 96.00 cycles   Mean Response Time : 76.81 cycles
 428:            Msg[29] Waiting  Time : 0.00 cycles   Mean Waiting  Time : 2.13 cycles

11 matches in 1 file
```

Figure 3-4 Snapshot of a Simulation report

By analysing the results, we see that the message arrives to the network at *ta*= 397.02 cycles in node [2][2] and it reaches its destination which is the node [0][1] at *td* = 493.02 cycles.

So, the response time of the message is calculated *RT = td – ta = 493.02 – 397.02 = 96 cycles*.

The message during its lifecycle passes by 2 intermediate nodes which are [1][2] and [0][2] and making a total of 3 hops in order to reach the destination.

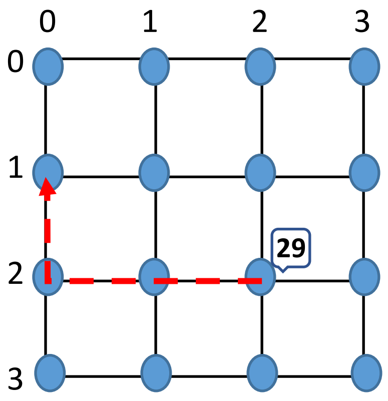


Figure 3-5 Demonstration of the path traversed by Msg [29]

According to [12], in the 2D mesh topology under low traffic the response time for a message can be given by*:*

$$RT = T_0 = H\left(tr + \frac{L}{b}\right)$$

Where:

- T_0 : the *Zero load latency*.
- H: Hop count, the number of channels traversed along a path from source to destination
- L: Message length = 32 phits.
- tr: Router delay, the delay through a single router. (= 0 negligeable)
- *b*: Channel bandwidth (1 phit / cycle)

So, a message that make 3 Hops with 32 phits length will have a response time

RT= H x L/b = 3 x 32/1 = 96 cycles.

And which is the case in the result measured by the simulation model.

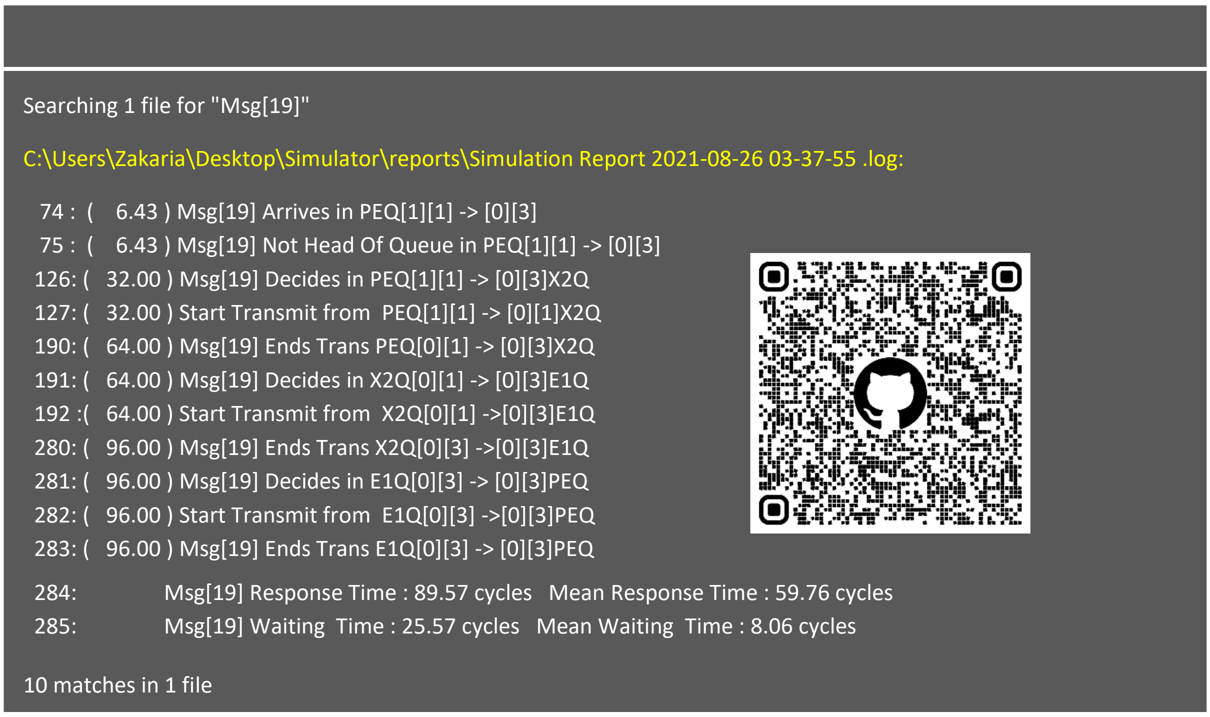
3.5.2 Run Complex Cases

In the previous section, we have tested the simulation model under simplified cases, where the message was transmitted directly without waiting in queues & buffers, this is because the links where free and the network didn't reach a saturation level yet. In this section we will analyse the behaviour of the simulation model under a heavy traffic that fellow the hotspot pattern (the hotspot node was selected to be in the centre of the network) and we will observe how the model performs when a certain saturation level is reached in some branches of the network especially in the neighbours of the hotspot node.

The simulation is performed under the following parameters:

- Topology: 4 x 4 Express Cube with 2 Hops
- Routing: Dimension-ordered (DOR)
- Traffic: Hotspot [2][2]
- Message length: 32 phits
- Injection Rate: 100
- Transmission Time: 32 Cycles

We open the generated log file and we track the lifecycle of the message with ID = 19.



```
Searching 1 file for "Msg[19]"

C:\Users\Zakaria\Desktop\Simulator\reports\Simulation Report 2021-08-26 03-37-55 .log:

  74 :  (   6.43 ) Msg[19] Arrives in PEQ[1][1] -> [0][3]
  75 :  (   6.43 ) Msg[19] Not Head Of Queue in PEQ[1][1] -> [0][3]
 126: (  32.00 ) Msg[19] Decides in PEQ[1][1] -> [0][3]X2Q
 127: (  32.00 ) Start Transmit from  PEQ[1][1] -> [0][1]X2Q
 190: (  64.00 ) Msg[19] Ends Trans PEQ[0][1] -> [0][3]X2Q
 191: (  64.00 ) Msg[19] Decides in X2Q[0][1] -> [0][3]E1Q
 192 :(  64.00 ) Start Transmit from  X2Q[0][1] ->[0][3]E1Q
 280: (  96.00 ) Msg[19] Ends Trans X2Q[0][3] ->[0][3]E1Q
 281: (  96.00 ) Msg[19] Decides in E1Q[0][3] -> [0][3]PEQ
 282: (  96.00 ) Start Transmit from  E1Q[0][3] ->[0][3]PEQ
 283: (  96.00 ) Msg[19] Ends Trans E1Q[0][3] -> [0][3]PEQ

 284:            Msg[19] Response Time : 89.57 cycles   Mean Response Time : 59.76 cycles
 285:            Msg[19] Waiting  Time : 25.57 cycles   Mean Waiting  Time : 8.06 cycles

10 matches in 1 file
```

Figure 3-6 Snapshot of a Simulation report

By analysing the results, we see that the message arrives to the network at *ta*= 6.43 cycles in node [1][1] and it reaches its destination which is the node [0][3] at *td*= 96.00 cycles.
So, the response time of the message is calculated *RT = td – ta = 96 – 6.43 = 89.57 cycles*.

The message during its lifecycle passes by an intermediate node which is [0][1] and making a total of 2 hops in order to reach the destination.

In the first hop in the node [1][1], the message has waited certain time *tw1 = 25.57 cycles* in the PE queue before being transmitted, due the occupation of the PE input by another message, this later frees the input at *t*=32 cycles at that time the message N°19 was placed as the head of queue and begin its transmission via the X- normal channel to the node [0][1].

Otherwise in the second hop and when the message arrives to the node [0][1], it traverses directly the E+ express channel, in this node there is no waiting time for the message because there was no message being transmitted by the node in this channel. So, *tw2 = 0 cycles*.

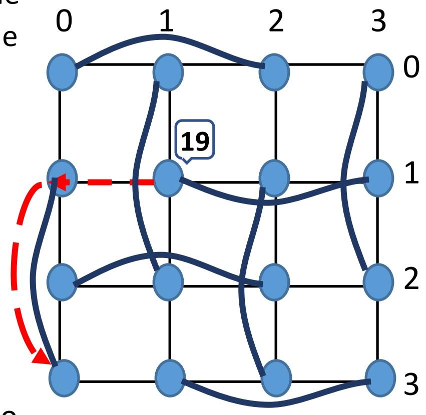


Figure 3-7 Demonstration of the path traversed by Msg [19]

The total waiting time of message N°19 while crossing the network is calculated as follows:

$$\text{Wait}_{Msg_{19}} = tw1 + tw2 = 25.57 \text{ cycles}$$

According to [19], The response time of message from its source to its destination is expressed as shown below:

$$RT_{Msg_i} = PDelay_{Msg_i} \times nbr_{hops} + (L-1) \times FT + \text{Wait}_{Msg_i}$$

Where:

- $PDelay_{Msg_i}$ is the physical delay, caused by the network physical aspects such as switch and link delay which equals to transmission time in our model (=32 cycles).
- nbr_{hops} is the number of routers traversed by the message which equals to 2 Hops in the example we take (one normal hop and one express hop).
- *FT* is the transmission time of one flit (=32 cycle).
- *L* is the message size in flits (*L*=1).
- Wait_{pkt_i} is the contention delay defined as the buffering time.

$$RT_{Msg_{19}} = PDelay_{Msg_{19}} \times nbr_{hops} + (L-1) \times FT + \text{Wait}_{Msg_{19}}$$

$$RT_{Msg_{19}} = 32 \times 2 + (1-1) \times 32 + 25.57$$

$$RT_{Msg_{19}} = 89.57 \; cycles$$

And which is the case in the result measured by the simulation model.

The comparison between the calculated result and the results obtained from the simulation both in simplified and complex cases shows how much the implemented model is near and representative to the real systems and proves the validity of the simulation model and how it correctly implements the assumptions.

Chapter 4 : Performance evaluation between 2D Mesh and Express Cube topologies

In this chapter, we will use the simulation model described in the previous chapter to carry out the performance comparison between the most preferred topology due to its regularity and scalability namely the two-dimensional (2D) mesh and the express cube topology with its variations of 2 Hops, 4 Hops and 8 Hops on the other hand.

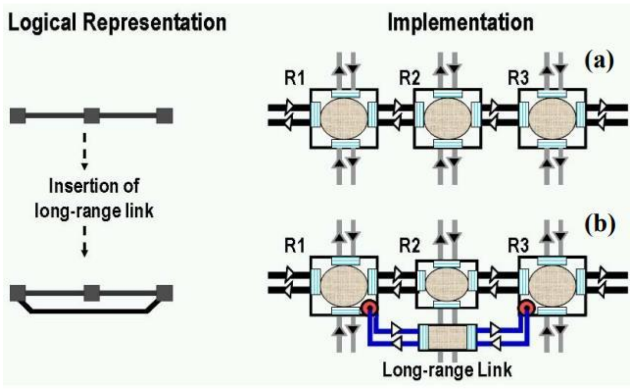
In the first stage, the comparison is carried out assuming no technological constraints imposed on the system implementation. The four topologies are compared in their 2D version. Both the uniform traffic and the hotspot traffic patterns have been considered in the comparison. In the second stage, and in order to maintain a fair performance comparison between these topologies, the physical constraints imposed by the implementation technology were taken into consideration, notably the bisection width which is relevant to the implementation of NoCs, for more detail see section 7.1.5 in [3].

In what follows, we will start by outlining the assumptions used in this study, then describing the method for collecting the simulation results, then discussing the simulation model validation. 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 been adopted in existing studies [14, 20, 10]:

- Message generation at a node is independent of all other nodes.
- The message arrival rate at each node follows a Poisson distribution with a mean interarrival rate 1/λ messages/cycle. Thus, the message inter-arrival time follows an exponential distribution with a mean arrival time λ cycles.
- The generated messages are of fixed length (L= 32 phits).
- Routing time (time for a router to decide which output to select for a given message) is negligible.
- Propagation delay across the links is negligible, the same thing across the express channels of the Express cube, this is possible due to the theory of latency-insensitive design that allows to increase the robustness of a design implementation, and which consists that any delay variations of a channel can be "recovered" by changing the channel latency while the overall system functionality remains unaffected. [21]



This can be achieved by segmenting Express channels into regular, fixed length, network channel connected by repeaters as demonstrated in *Figure 4-1* and explained in [22].

Figure 4-1 Implementation of the repeaters. Routers 1 and 3 are both connected by Router 2 (a), the underlying mesh network, and the inserted long-range link (b) [22].

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 performance comparison studies [10, 13, 15].

- **Uniform:** Each node has an equal probability of sending a message to any other node.
- **Hotspot:** The Nodes favour 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, 16x16 and 32x32 nodes.

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 as shown in (1).

$$RT_i = T_{now} - T_{arrival} \text{.........} (1)$$

Where:

- T_{now} is the time at which the message *i* reaches the destination node.
- $T_{arrival}$ is the time of generation of the message *i* 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 shown in (2).

$$RT = \frac{\sum_{i=1}^{max} RT_i}{max} \text{............} (2)$$

Where: RT_i is the response time of an individual message i and max is the number of all delivered messages in the network during the simulation.

4.3.2 Mean waiting time

The waiting time is also a qualitative measure of network performance. The waiting time for a single message is the time spent in the queues all along the nodes traversed from the source node until the destination node. The waiting time of message *i* is also measured in number of cycles expressed in (3).

$$WT_i = \sum_{j=1}^{n} Tw_j \dots\dots\dots (3)$$

Where:

- Tw_j : is the time spent by a message waiting in the queue in node *j*
- n : is the number of nodes traversed by the message

The mean waiting time is then found by averaging the waiting time over all delivered messages. This can be written as shown in (4).

$$WT = \frac{\sum_{i=1}^{max} WT_i}{max} \dots\dots\dots (4)$$

Where:

- WTi : is the waiting time of an individual message i
- max : is the number of all delivered messages in the network during the simulation.

4.3.3 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 as shown in (5).

$$Th = \frac{\text{Total messages delivered}}{\text{Simulation time}} \dots\dots\dots (5)$$

4.4 Method to collect the simulation results

The method used during the simulation to collect the results known as the batch means method, also called the method of subsamples, it consists of running a long simulation run, discarding the initial transient interval, and dividing the remaining observations run into several batches or subsamples [16]. It was selected due its popularity to its simplicity and effectiveness.

In our case of study and for each simulation scenario, a total of 20 000 messages were generated and delivered to their destination, those 20 000 messages were divided into 10 subsamples or batches, as a result the batch size is equal to 2000 messages. then we proceed as follows:

4.4.1 Compute means for each batch

For each batch, the mean response time, the mean waiting time and the throughput are calculated from the 2000 messages by the equations mentioned in the previous section.

4.4.2 Compute the overall mean

is the mean of the means of the batches. In the example below the overall mean of the waiting time is calculated as shown in (6).

$$OMWT = \frac{1}{m}\sum_{i=1}^{m} WT_{bi} \ldots\ldots\ldots (6)$$

Where:

- $OMWT$: is the overall mean of the waiting time
- WT_{bi} : is the mean waiting time for the batch with number *i*
- m : is the number of batches used during the simulation

4.4.3 Calculate the dispersion of batch means

Sometimes the mean or the average alone is not sufficient to represent a collection of data especially if there is a large variability between the values of the data of the collection. The variability is commonly specified by the variance. It is denoted by *s^2* and is computed as expressed in (7).

$$s^2 = Var(WT) = \frac{1}{m-1}\sum_{i=1}^{m}(WT_{bi} - OMWT)^2 \ldots\ldots\ldots (7)$$

Note that the variance is written as *s^2*, it has squared units. In our example, the variance of the waiting time measured in *cycles* will be given in *cycles squared*.
Since the variance of the waiting time is a squared quantity, it cannot be directly compared to the waiting time values or to the mean waiting time value of a set of messages. It is therefore more useful to have a quantity which is the square root of the variance. This quantity is known as the *standard deviation*, and it is more meaningful because it is expressed in the same unit as the mean value. It is denoted by *SD* and is calculated as expressed in (8).

$$SDWT = \sqrt[2]{Var(WT)} \ldots\ldots\ldots (8)$$

4.4.4 Calculate the confidence interval for the mean

A confidence interval provides information about the possible range of values for the performance measure. A narrow confidence interval indicates that the performance measure has been estimated with a high degree of precision. A wide confidence interval, on the other hand, indicates that the precision is not high.

Knowing the precision is often more helpful to analyse the simulation results, in our case of study and in order to calculate the confidence interval of the waiting time mean as an example, we use a function categorized under the Excel Statistical functions and which is the

Confidence Interval function, it will use the normal distribution to calculate and return the margin of error for the means, the margin of error for the waiting time means it’s given as shown in (9).

$$MEWT = CONFIDENCE(\alpha, SDWT, m) \dots\dots\dots (9)$$

Where:

- $MEWT$: is the margin of error for the waiting time means
- α: is the significance level. The significance level is equal to 1– confidence level. So, a significance level of 0.05 is equal to a 95% confidence level.
- m : is the number of batches used during the simulation (number of means)

The confidence interval gives us the actual low and high limits of the waiting time mean at a given significance level α. These limits are one *MEWT* below the waiting time mean and one *ME* above it, Notice the use of "at a given significance level α" If we want to be surer that the unknown value is within one *MEWT* of the waiting mean, we need a better significance level.

The value of α is equal to *0.05* and is a common one; it means there's only a *5%* chance our confidence interval will not capture the true value. Using α=0.01 would mean there's only a 1% chance. Of course, there's a trade-off. If we want increased confidence, we have to take a wider interval. The Excel function above for a significance level of *5%*, it will calculate the confidence interval for the waiting time as expressed in (10).

$$\overline{WT} \pm 1.96 \frac{SDWT}{\sqrt{m}} \dots\dots\dots (10)$$

The use of the excel function instead of integrating this formula in the simulator is more preferable, because it gives the analyst or the examiner of the simulation results the ability to choose the significance level α while interpreting and analysing the results.

4.5 Results and Discussion

In this section we present and analyse the simulation results by comparing the topologies under both unconstrained and constrained implementations.

4.5.1 Mesh vs Express Cube (2 Hops) vs Express Cube (4 Hops) vs Express Cube (8 Hops): Unconstrained implementation

In this case, the topologies are not be subjected to any physical constraints imposed by the 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 2D Mesh and the Express cube with its variations as well are subjected to various traffic injection rates (λ) using the uniform traffic pattern, where a sender node has an equal probability of sending a message to any other destination node.

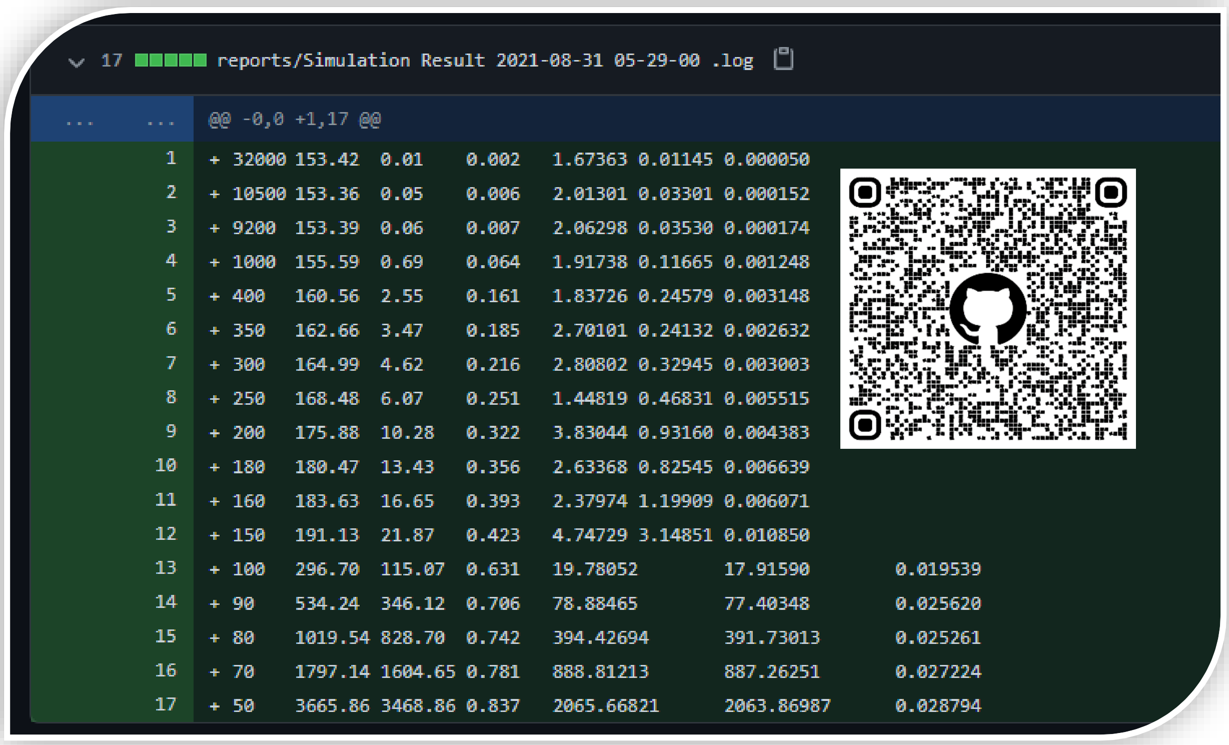
Before presenting the performance results for the topologies, we show first in *Table 4-1* an example of the performance and the statistical results obtained from a simulation of the 2D mesh topology with 8x8 nodes, this table generates and draws the blue curves (2D Mesh) with the error bars in the graphs in *Figure 4-3*, in what follows all the graphs are generated from data provided by tables like *Table 4-1,* where each table is associated to a curve specifically to a topology from one simulation.

Notice that you will find the associated tables below and theirs graphs of all the performed simulation during our study in the Excel file attached to the report.

These tables are generated by the report generator module during the simulation and they are stored in structured and formatted files inside the "reports" folder as shown in *Figure 4-2* in order to facility the collection and the exportation of the results to the Excel file.
Each performance result in *Table 4-1* and in all the figures below has been collected from 10 batches where each batch reflects the statistics of at least 2000 delivered messages.

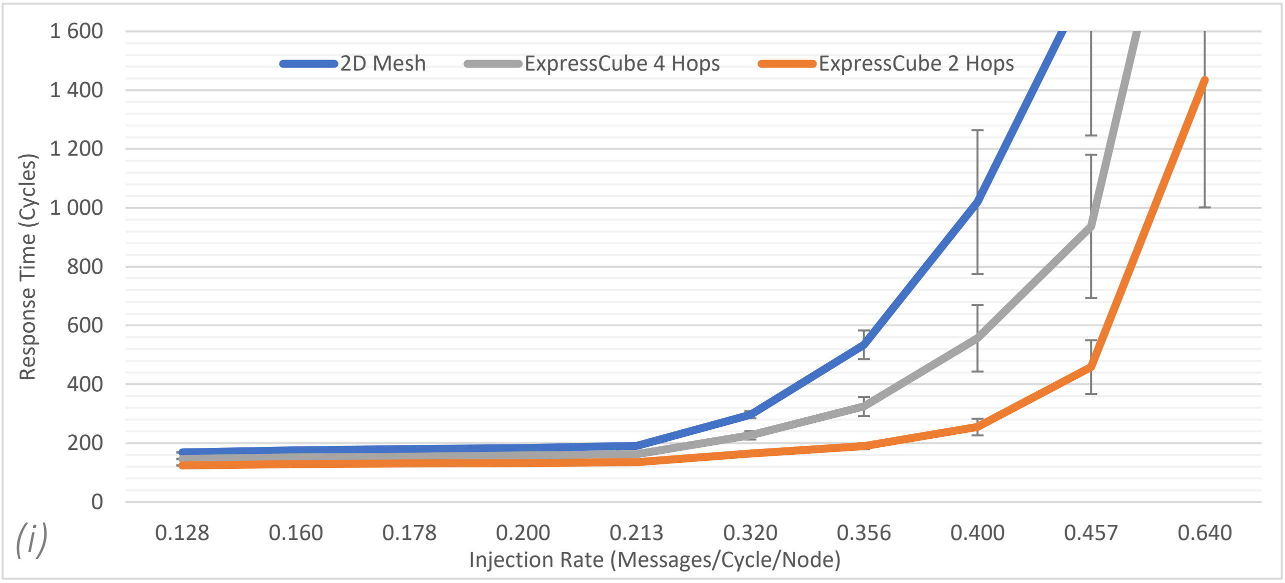
λ	RT	WT	TH	SDRT	SDWT	SDTH	MERT	MEWT	METH
32000	153.42	0.01	0.002	1.67363	0.01145	0.00005	1.03730756	0.007097	3.1E-05
10500	153.36	0.05	0.006	2.01301	0.03301	0.000152	1.247653598	0.020459	9.42E-05
9200	153.39	0.06	0.007	2.06298	0.0353	0.000174	1.278624756	0.021879	0.000108
1000	155.59	0.69	0.064	1.91738	0.11665	0.001248	1.188382599	0.072299	0.000774
400	160.56	2.550	0.161	1.83726	0.24579	0.003148	1.138724621	0.152339	0.001951
350	162.66	3.47	0.19	2.70	0.24	0.003	1.674	0.150	0.001631
300	164.99	4.62	0.216	2.80802	0.32945	0.003003	1.740396847	0.204191	0.001861
250	168.48	6.07	0.251	1.44819	0.46831	0.005515	0.897580968	0.290256	0.003418
200	175.88	10.28	0.322	3.83044	0.9316	0.004383	2.374087684	0.577401	0.002717
180	180.47	13.43	0.356	2.63368	0.82545	0.006639	1.632341781	0.51161	0.004115
160	183.63	16.65	0.393	2.37974	1.19909	0.006071	1.47495103	0.74319	0.003763
150	191.13	21.87	0.42	4.75	3.15	0.01085	2.942346759	1.951431	0.006725
100	296.70	115.07	0.631	19.78052	17.9159	0.019539	12.25986803	11.10419	0.01211
90	534.24	346.12	0.71	78.88	77.40	0.02562	48.8923142	47.97429	0.015879
80	1019.54	828.7	0.742	394.42694	391.73013	0.025261	244.463858	242.7924	0.015657
70	1797.14	1604.65	0.781	888.81213	887.26251	0.027224	550.8813428	549.9209	0.016873
50	3665.86	3468.86	0.837	2065.6682	2063.8698	0.028794	1280.290895	1279.176	0.017846

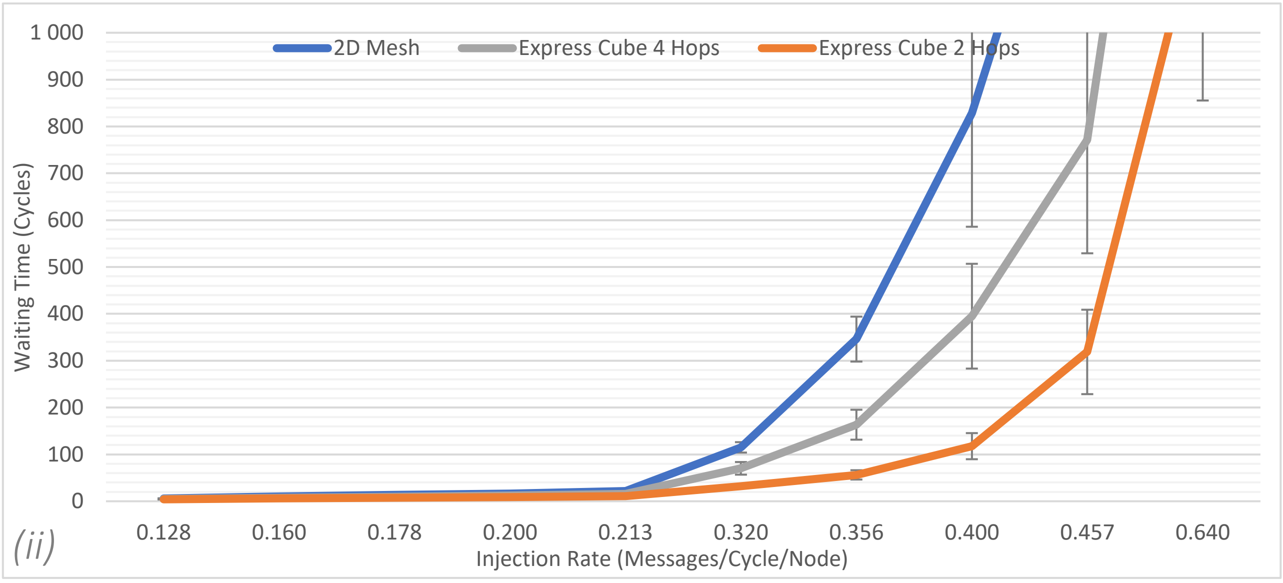
Table 4-1 Results for Simulation of the 2D mesh topology with 8x8 nodes



```
17 reports/Simulation Result 2021-08-31 05-29-00 .log
@@ -0,0 +1,17 @@
+ 32000 153.42  0.01    0.002   1.67363 0.01145 0.000050
+ 10500 153.36  0.05    0.006   2.01301 0.03301 0.000152
+ 9200  153.39  0.06    0.007   2.06298 0.03530 0.000174
+ 1000  155.59  0.69    0.064   1.91738 0.11665 0.001248
+ 400   160.56  2.55    0.161   1.83726 0.24579 0.003148
+ 350   162.66  3.47    0.185   2.70101 0.24132 0.002632
+ 300   164.99  4.62    0.216   2.80802 0.32945 0.003003
+ 250   168.48  6.07    0.251   1.44819 0.46831 0.005515
+ 200   175.88  10.28   0.322   3.83044 0.93160 0.004383
+ 180   180.47  13.43   0.356   2.63368 0.82545 0.006639
+ 160   183.63  16.65   0.393   2.37974 1.19909 0.006071
+ 150   191.13  21.87   0.423   4.74729 3.14851 0.010850
+ 100   296.70  115.07  0.631   19.78052        17.91590        0.019539
+ 90    534.24  346.12  0.706   78.88465        77.40348        0.025620
+ 80    1019.54 828.70  0.742   394.42694       391.73013       0.025261
+ 70    1797.14 1604.65 0.781   888.81213       887.26251       0.027224
+ 50    3665.86 3468.86 0.837   2065.66821      2063.86987      0.028794
```

Figure 4-2 Simulation Result for the 2D Mesh 8x8 nodes





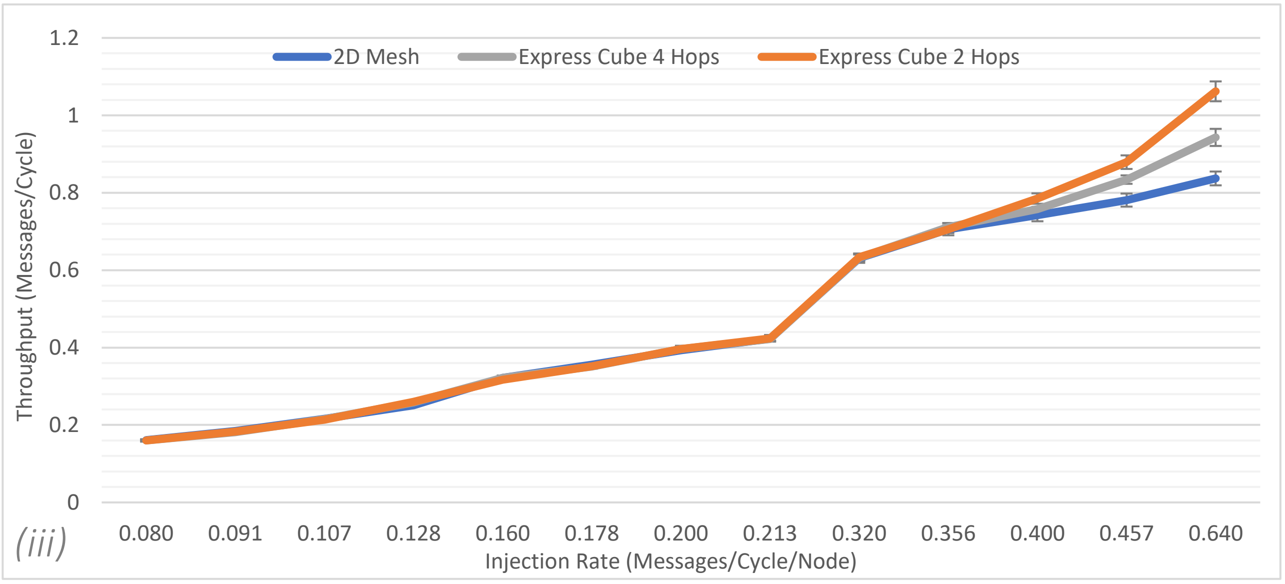


Figure 4-3 Performance results for the 2D Mesh vs EC2H vs EC4H under unconstrained uniform traffic for 8x8 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

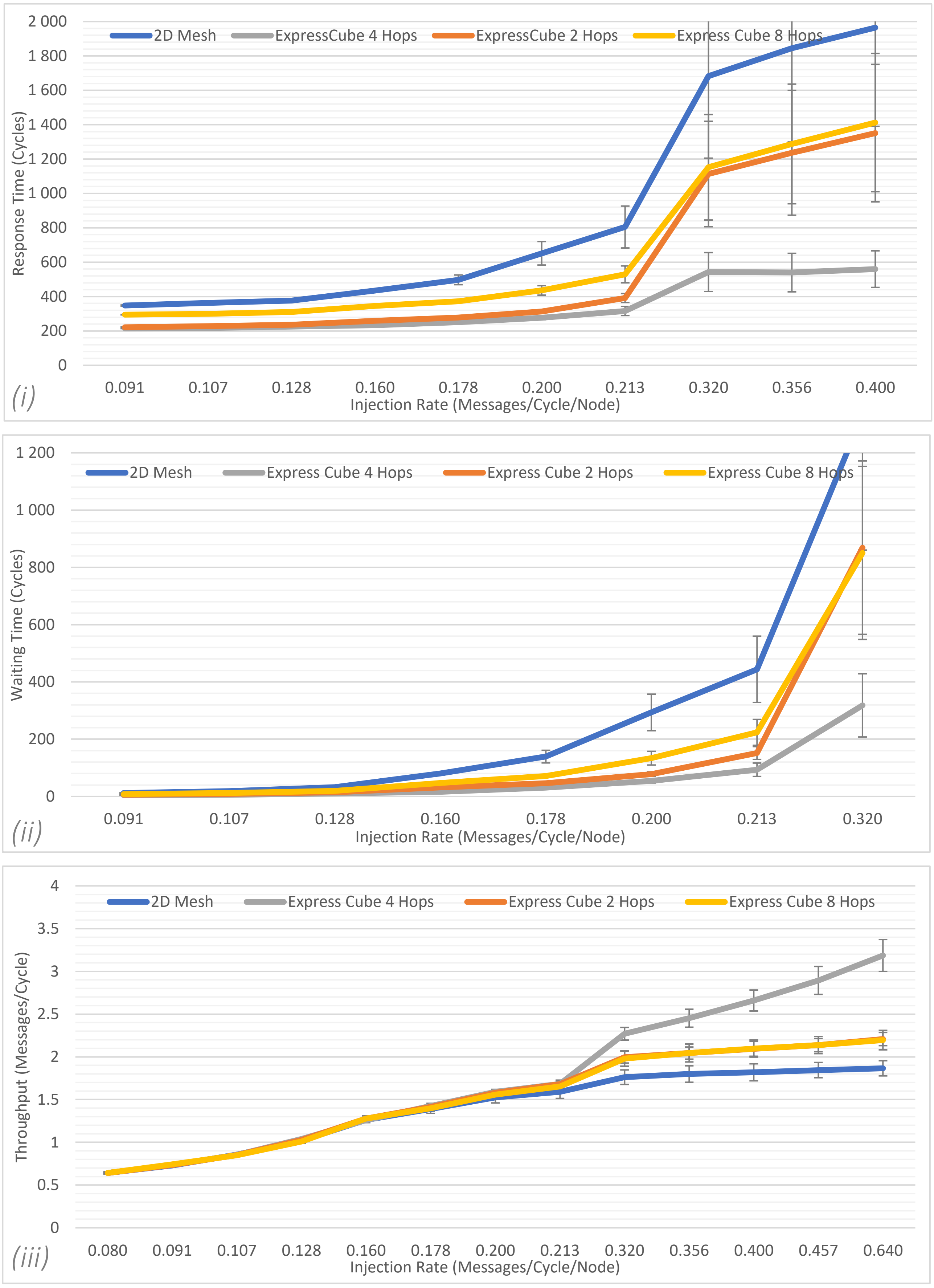


Figure 4-4 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under unconstrained uniform traffic for 16x16 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

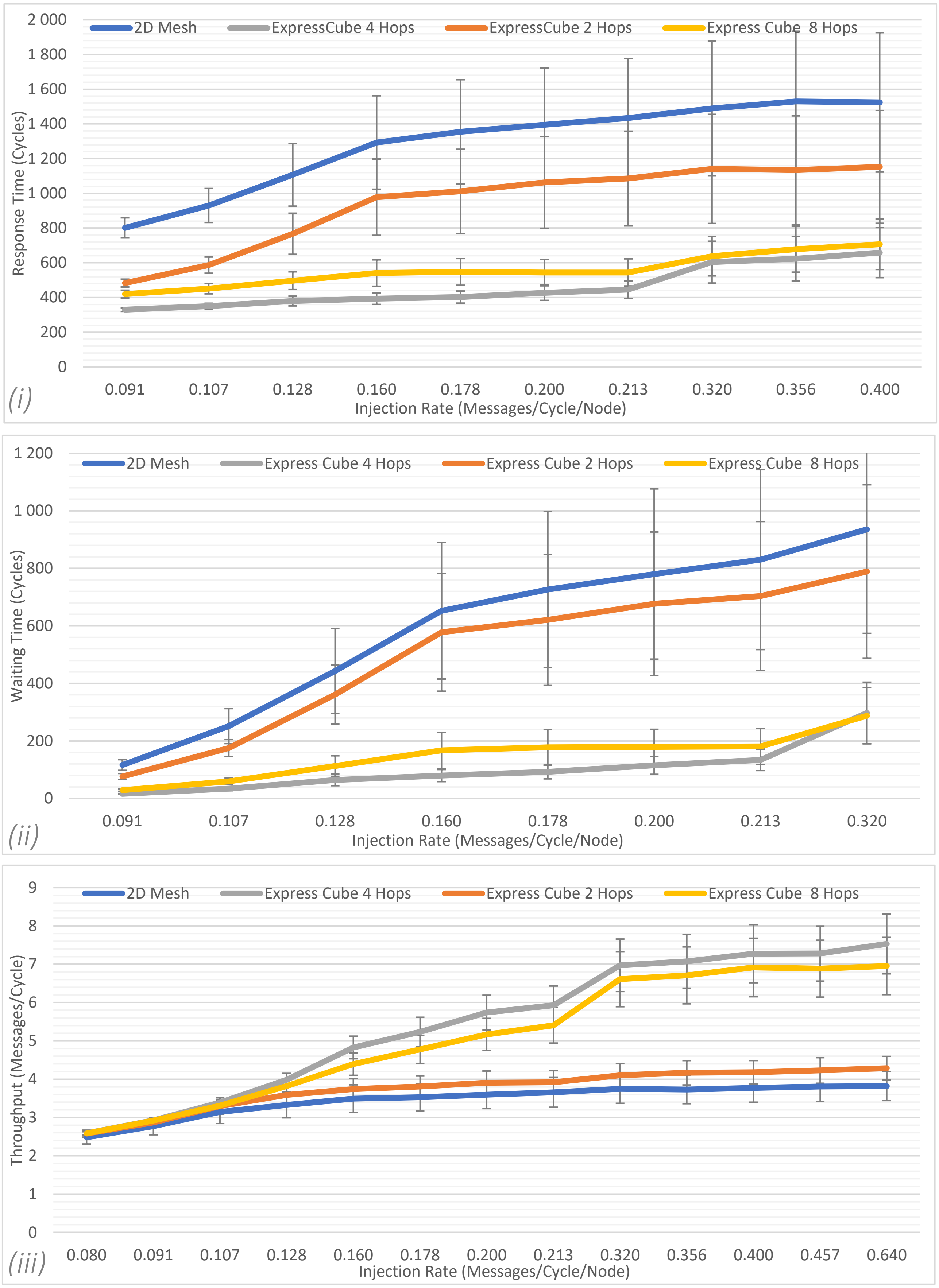


Figure 4-5 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under unconstrained uniform traffic for 32x32 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

The four Network-On-Chip topologies are compared amongst each other.
The simulation results are depicted below for network sizes of 8x8 ,16x16 and 32x32 nodes.

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 *Figure 4-3(i)* to *Figure 4-16(i)* the y-axis represents the mean message response time (Measured in cycles) and in figures *Figure 4-3(ii)* to *Figure 4-16(ii)* the y-axis represents the mean message waiting time (Measured in cycles) while in figures *Figure 4-3(iii)* to *Figure 4-16(iii)* the y-axis represents the mean throughput (Measured in messages/cycle).

The *Figure 4-3* shows that the EC2H has a lowest response time and waiting time compared to the mesh and the EC4H under all levels of injection rates and an equal throughput with slight superiority can be observed under an intensive traffic rate, that because it has more express channels than the other topologies and the chance for a message to pass by 2-Hops channels is superior then passing by normal links in the mesh or by the 4-hops channels in the EC4H.

The *Figure 4-4* reveals that the EC4H topology has the best performance among the others, while the EC2H and the EC8H show similar behaviours under different load traffic injections, this can be explained by the fact that the bypassing of nodes in the EC4H reduces effectively the diameter and the average distance and balance between the local and the non-local messages crossing the topology.

The *Figure 4-5* exhibit also the effectiveness of the EC4H, unlike the previous figure we see here that the EC8H performs better too, this can be explained by the good exploitation and utilization of the 8-Hops Channels by the long-range messages.

Regardless of the network size or load traffic injection, the mesh topology shows the worst performances compared to the Express cube topologies, this can be interpreted by its long diameter, which results in larger amounts of traffic congestions all along the paths crossed by messages from their sources until they reach their destinations.

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.

The *Figure 4-6* to *Figure 4-8* are the performance results for network sizes of 8x8, 16x16 and 32x32 nodes under unconstrained implementation where α is set to *0.1*.
The figures reveal that the same conclusions as scenario 1 are reached in that the EC2H exhibits superior performance in the 8x8 network size, while in the 16x16 and 32x32 topologies the EC4H performs better. This is due to the effective bypassing and to its lower average message distance.

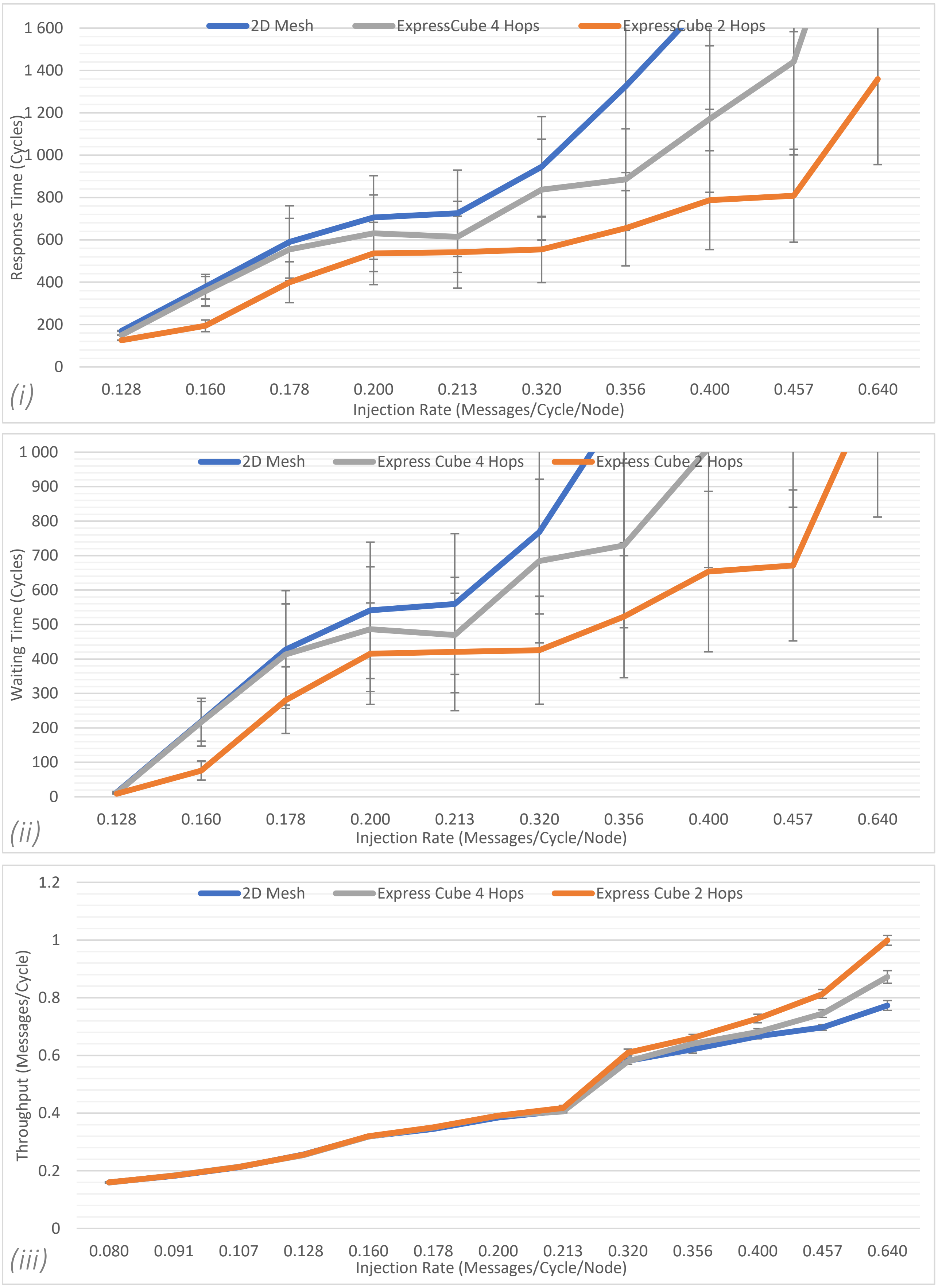


Figure 4-6 Performance results for the 2D Mesh vs EC2H vs EC4H under unconstrained hotspot traffic for 8x8 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

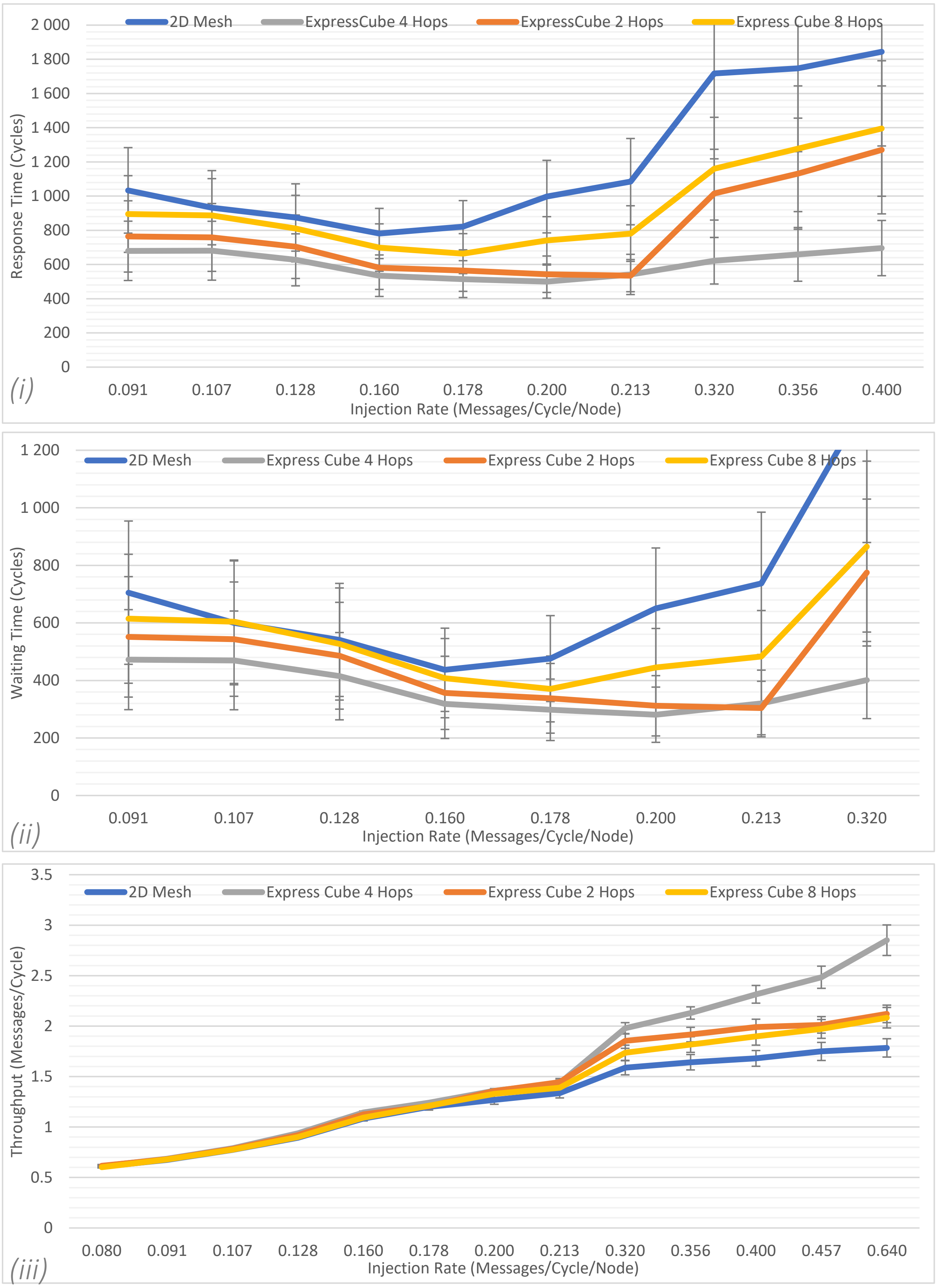


Figure 4-7 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under unconstrained hotspot traffic for 16x16 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

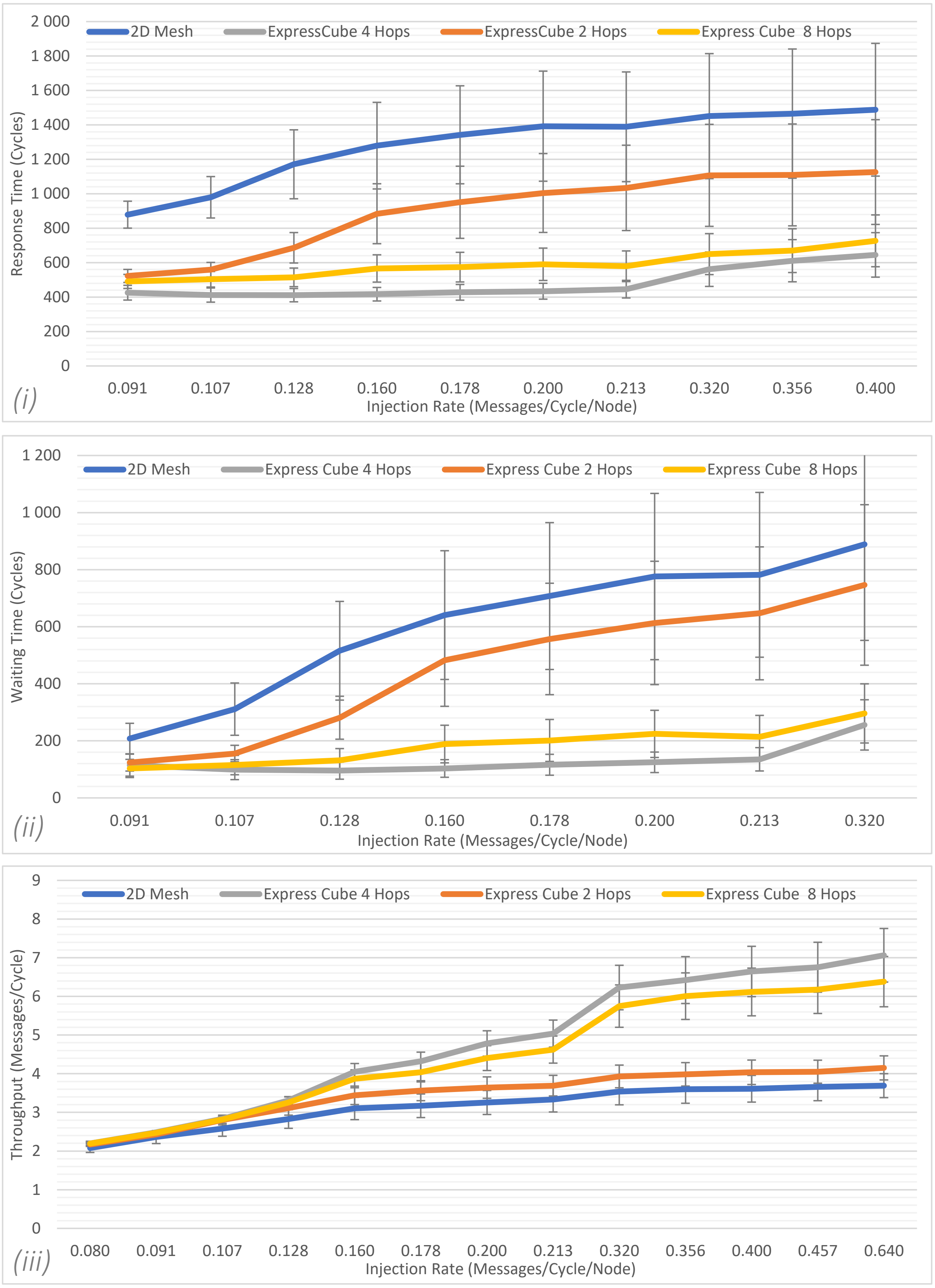


Figure 4-8 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under unconstrained hotspot traffic for 32x32 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

4.5.2 Mesh vs Express Cube (2 Hops) vs Express Cube (4 Hops) vs Express Cube (8 Hops): Constrained implementation

In the previous section, the performances were evaluated and measured based on the graphical properties only without taking into account any physical or hardware limitations. However, in this section we will evaluate and present once more the performances of the Mesh and the Express Cube with its variations under the physical implementation constraints in order to conduct a fair comparison between the two topologies.

One of the dominant physical constraints in the implementation of interconnection networks is the available wiring area [3]. The available wiring space in the Express Cube NoC is more than in a mesh of the same size, that is due to the express channels. Furthermore, an Express Cube requires significant number of links which must be factored into the wiring budget. If the available wire tracks along the bisection is fixed, an Express Cube will be restricted to narrower links than a Mesh, thus lowering per-link bandwidth, and increasing transmission delay.

These contrasting properties illustrate the importance of considering implementation details in evaluating and analysing the performances of the topologies.

The bisection width of the mesh topology as mentioned in [23] is given as expressed in (11).

$$B_{mesh} = \mathbf{2\sqrt{N}}\boldsymbol{k}^{\frac{n}{2}-1}\,\mathbf{W_{mesh}} \,\ldots\ldots\ldots (11)$$

Where:

- $\boldsymbol{k}$: number of nodes per dimension
- $\boldsymbol{n}$: number of dimensions

The expression can be interpreted as follows: for each of the $\sqrt{N}$ rows of the network, there are $k^{((n/2)-1)}$ of these channels in each direction for a total of $2\sqrt{N}k^{((n/2)-1)}$ channels. Thus, the bisection width *Bmesh* of a k-ary n-cube with *Wmesh* wide communication channels is

$$Bmesh = 2\sqrt{N}k^{\frac{n}{2}-1} * \ Wmesh \quad [23]$$

In the case of the Express Cube topology, the same equation above is applied to calculate the bisection width expect that we need to add the Express channels that cross the midpoint of the network in each row, the number of these express channels is depending on the Number of hops *H*.

For example, in the EC2H we have one Express channel crossing the midpoint in any given row.

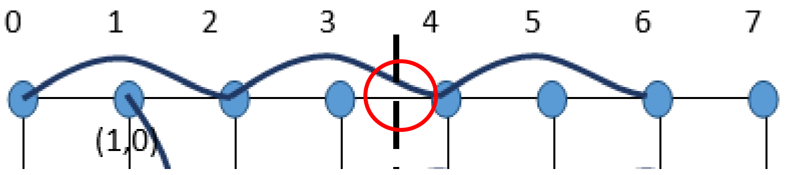


Figure 4-9 a Row of the EC2H

Following the same reasoning of Dally, it will be for each of the $\sqrt{N}$ rows of the network, there are **2**$k^{((n/2)-1)}$ of these channels in each direction for a total of $2\sqrt{N}\mathbf{2}k^{((n/2)-1)}$ channels. Thus, the bisection width B_{EC2H} of the Express Cube with 2 Hops with W_{EC2H} wide communication channels is $B_{EC2H} = 2\sqrt{N}2k^{\frac{n}{2}-1} * W_{EC2H}$

In the EC4H we have two Express Channels crossing the midpoint in any given row.

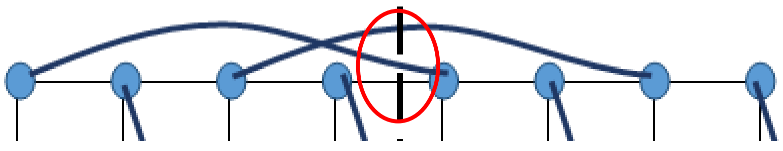


Figure 4-10 a Row of the EC4H

Following the same reasoning of Dally once more, it will be for each of the $\sqrt{N}$ rows of the network, there are **3**$k^{((n/2)-1)}$ of these channels in each direction for a total of $2\sqrt{N}3k^{((n/2)-1)}$ channels. Thus, the bisection width B_{EC4H} of the Express Cube with 4 Hops with W_{EC4H} wide communication channels is

$$B_{EC4H} = 2\sqrt{N}3k^{\frac{n}{2}-1} * W_{EC4H}$$

We Generalize the equation for any given *M* Hops (consider that *M* is an even number):

$$\boldsymbol{B_{ECMH} = 2\sqrt{N}\left(\frac{M}{2}+1\right)k^{\frac{n}{2}-1} * W_{ECMH}}$$

By applying the Bisection Bandwidth Constraint, the limited available wiring area is represented by a fixed bisection width which will give us:

$$B_{ECMH} = B_{Mesh}$$

$$2\sqrt{N}\left(\frac{M}{2}+1\right)k^{\frac{n}{2}-1} * W_{ECMH} = 2\sqrt{N}\,k^{\frac{n}{2}-1} * W_{mesh}$$

Therefore, the channel width of a link in the Express Cube with *M* Hops W_{ECMH} can be expressed in terms of that of the mesh as

$$(\frac{M}{2}+1)\,W_{ECMH} = W_{mesh}$$

$$\boldsymbol{W_{ECMH} = \frac{2}{M+2}\,Wmesh}$$

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 Express Cube with *M* Hops is less of that of the mesh. As a result, the message length in Express Cube with *M* Hops will be greater than the message length in the mesh topology.

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:

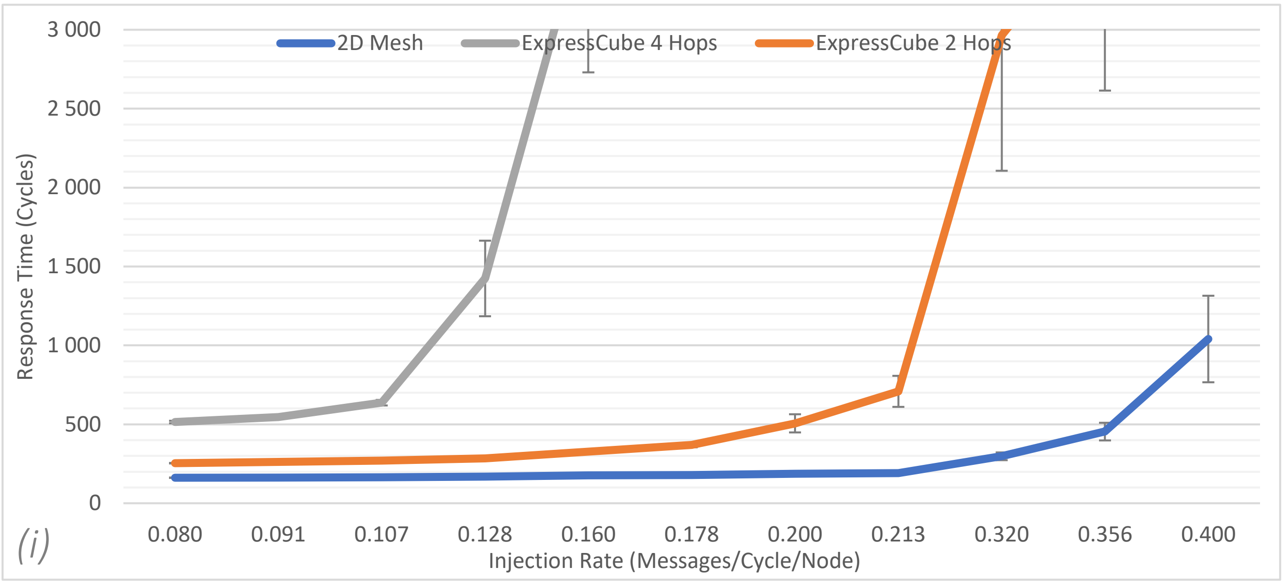
It will take 64 cycles in the EC2H Because $W_{EC2H} = \frac{2}{2+2}\text{Wmesh} = \frac{\mathbf{1}}{\mathbf{2}}\text{Wmesh}$

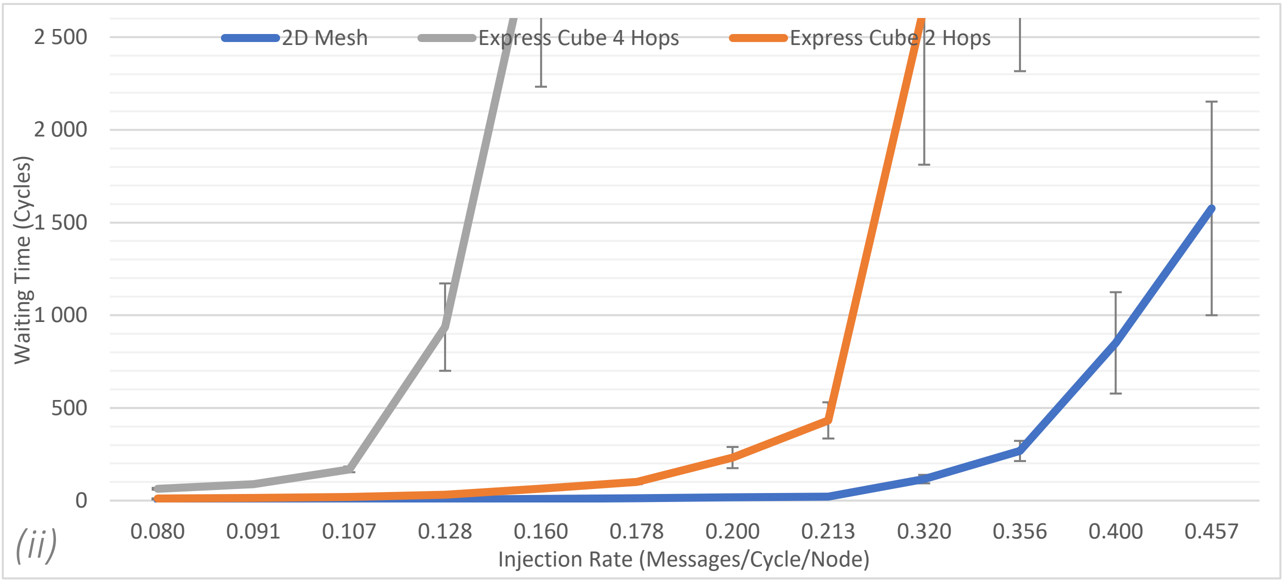
It will take 96 cycles in the EC4H Because $W_{EC4H} = \frac{2}{4+2}\text{Wmesh} = \frac{\mathbf{1}}{\mathbf{3}}\text{Wmesh}$

It will take 160 cycles in the EC8H Because $W_{EC8H} = \frac{2}{8+2}\text{Wmesh} = \frac{\mathbf{1}}{\mathbf{5}}\text{Wmesh}$

Scenario 1: Uniform traffic

The four topologies are subjected to a uniform traffic pattern. The following figures depict the results of the comparisons.
In figures Figure 4-11 to Figure 4-13 , the impact of the reduction in the channel bandwidth of Express Cube topologies can be clearly observed as it is now outperformed by the Mesh across the considered network sizes in terms of response time, waiting time and throughput.
This is in contrast to the performance outcome of the previous scenarios where the Express Cube topologies exhibited the best performance out of the Mesh topology.
However, the EC8H shows the worst performance among the Express cube topologies that can be explained by the effect of its significant message length that impacts directly the transmission time and results longer response time and extended waiting time.
While the EC2H and EC4H reveal almost the same behaviour despite the difference in the channel bandwidth. This might be attributed to the difference in average message distance between the two topologies, the gain of the performance in the unconstrained implementation for the EC4H over the EC2H is decreased in this case because of its longer message transmission delay comparing to the Express cube with 2 Hops topology.





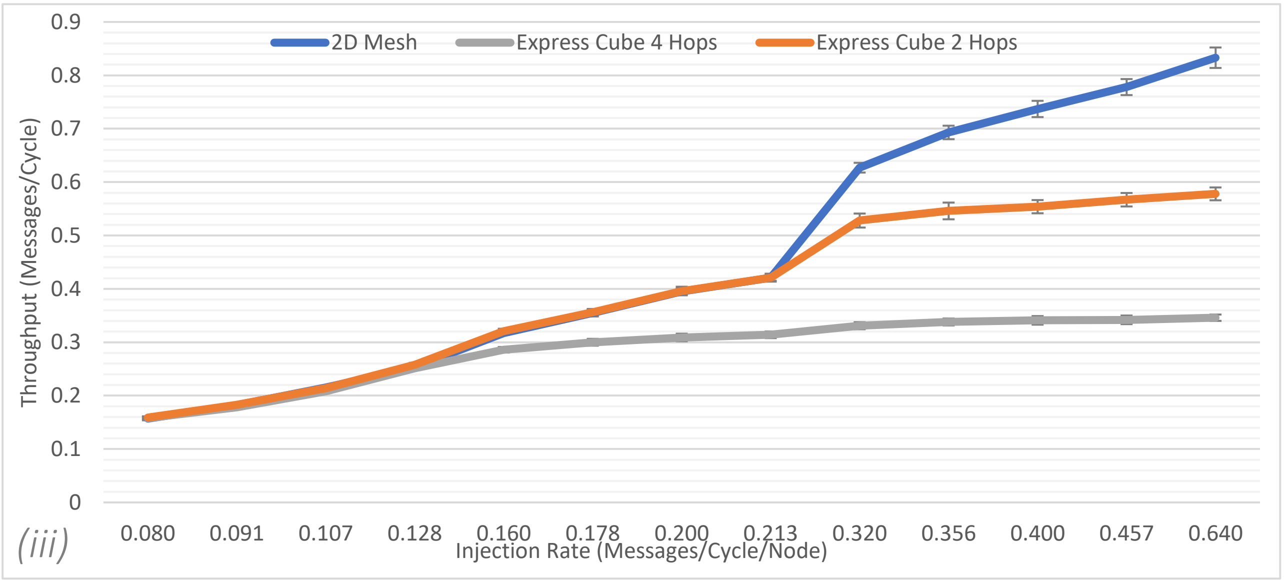


Figure 4-11 Performance results for the 2D Mesh vs EC2H vs EC4H under constrained uniform traffic for 8x8 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

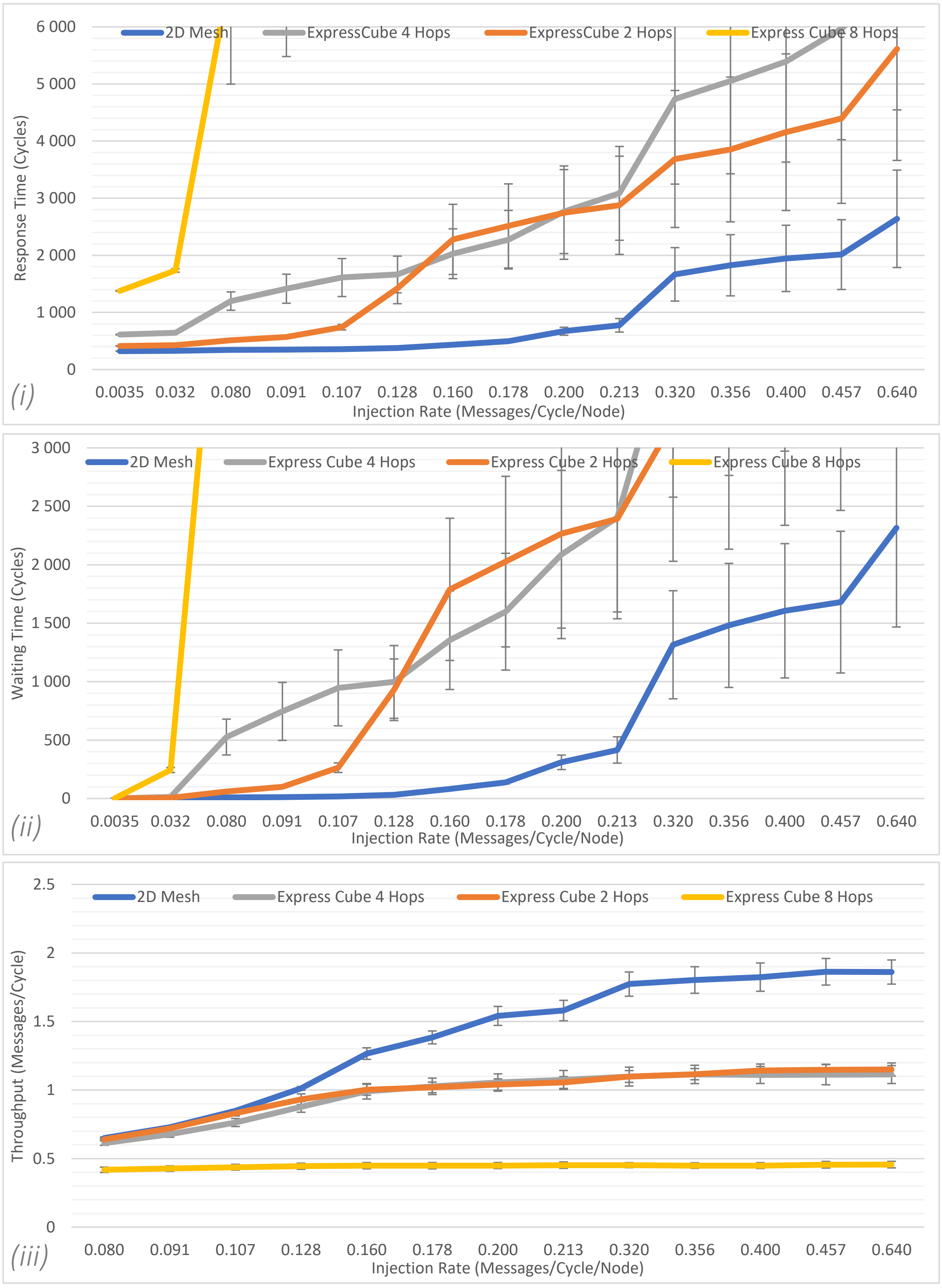


Figure 4-12 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under constrained uniform traffic for 16x16 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

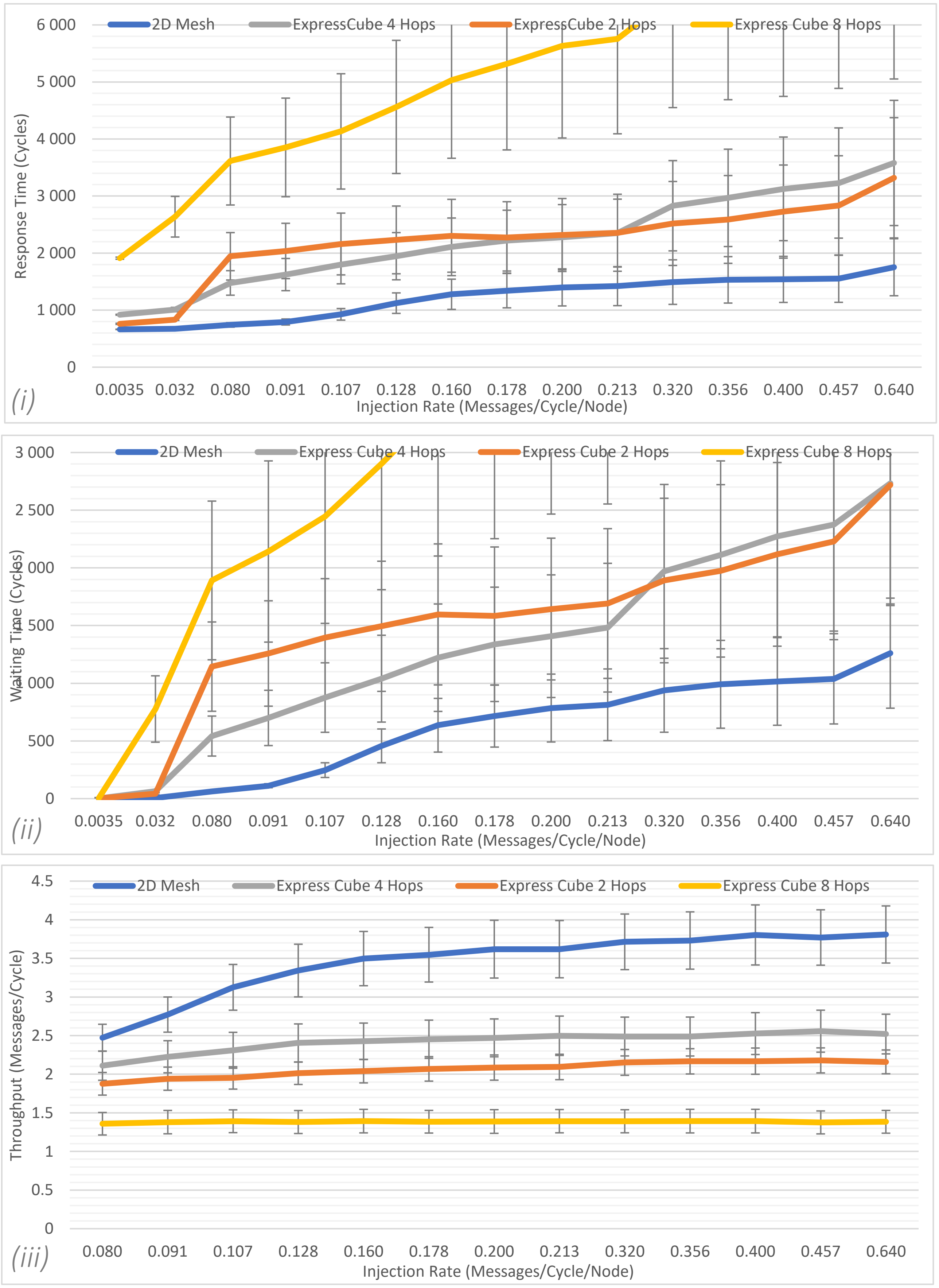


Figure 4-13 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under constrained uniform traffic for 32x32 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

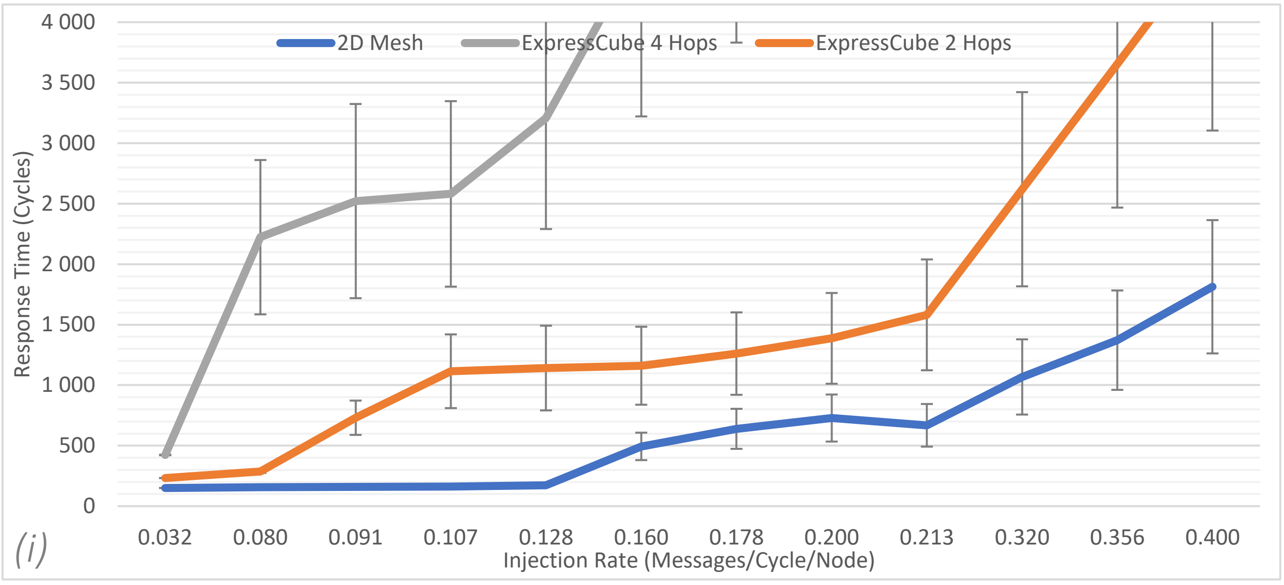
Scenario 2: Hotspot traffic

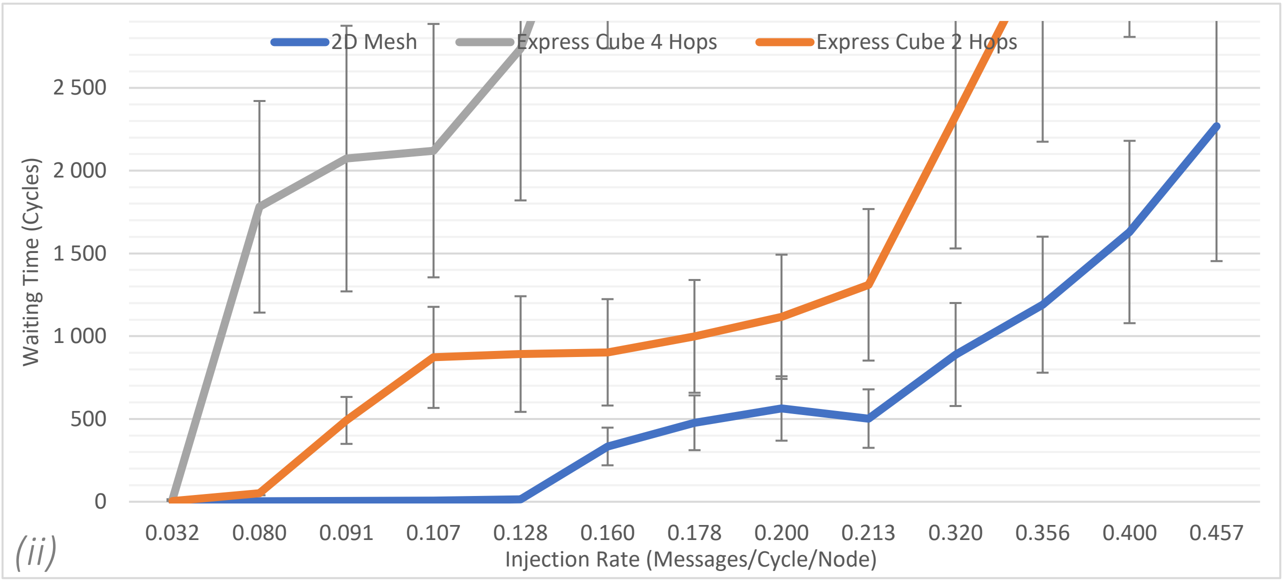
In *Figure 4-14* to *Figure 4-16*, similar conclusions can be drawn from the simulation results for all four topologies. That is, the 2D Mesh clearly outperforms the Express Cube topologies under different traffic injection loads for all of the performance measures.

The EC8H shows once more the worst performance among the Express Cube Topologies due to its significant message length that impacts directly the transmission time and results longer response time and extended waiting time.

Once again, The EC2H and EC4H reveal almost the same behaviour despite the difference in the channel bandwidth. This might be attributed to the difference in average message distance between the two topologies, the gain of the performance in the unconstrained implementation for the EC4H over the EC2H is decreased in this case because of its longer message transmission delay comparing to the Express cube with 2 Hops topology.

The conclusion that can be drawn from this is that despite the Express Cube 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.





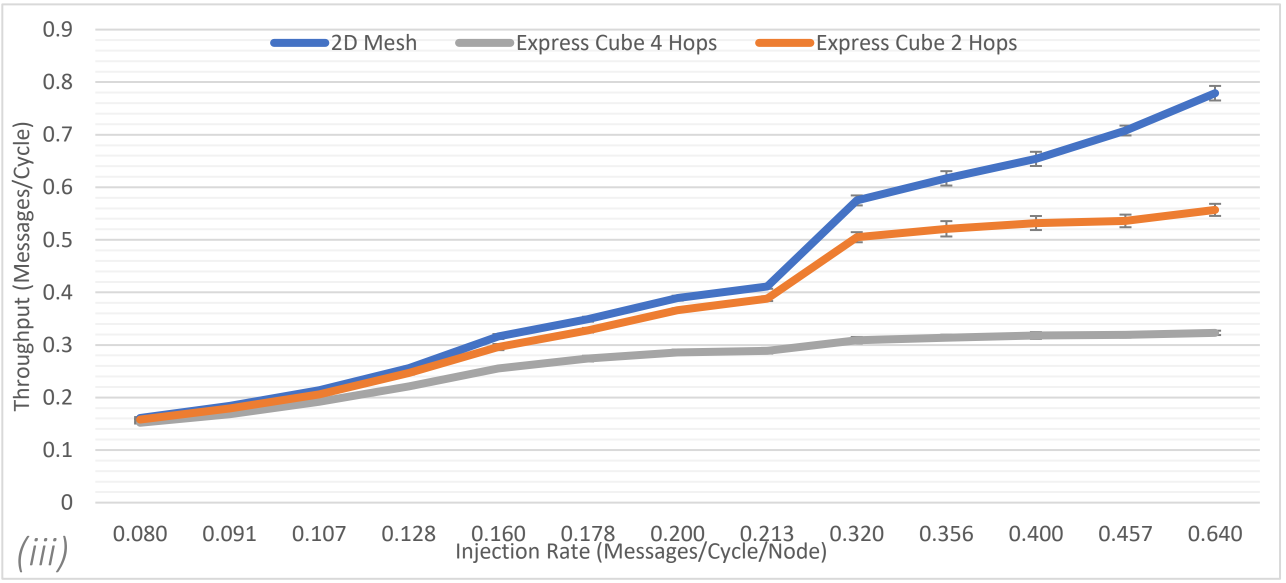


Figure 4-14 Performance results for the 2D Mesh vs EC2H vs EC4H under constrained hotspot traffic for 8x8 nodes (i) Response Time, (ii) Waiting Time, (iii) *Throughput*

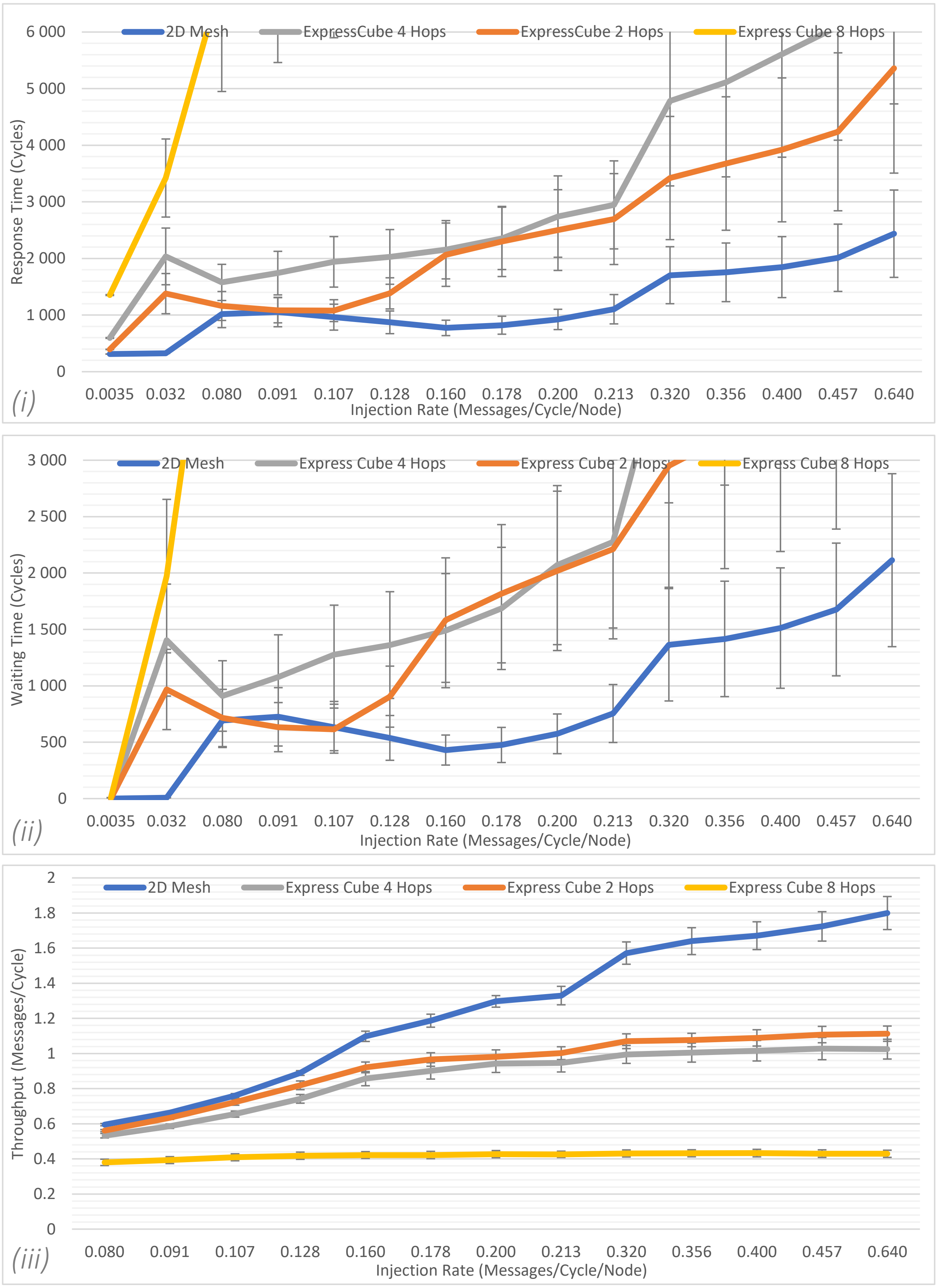
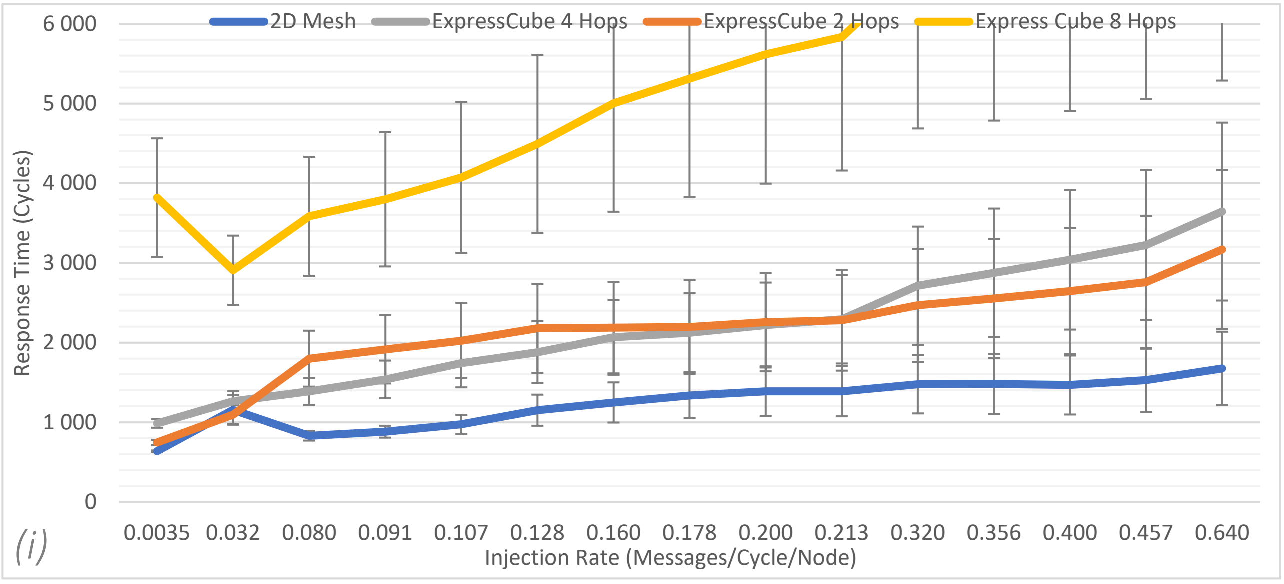
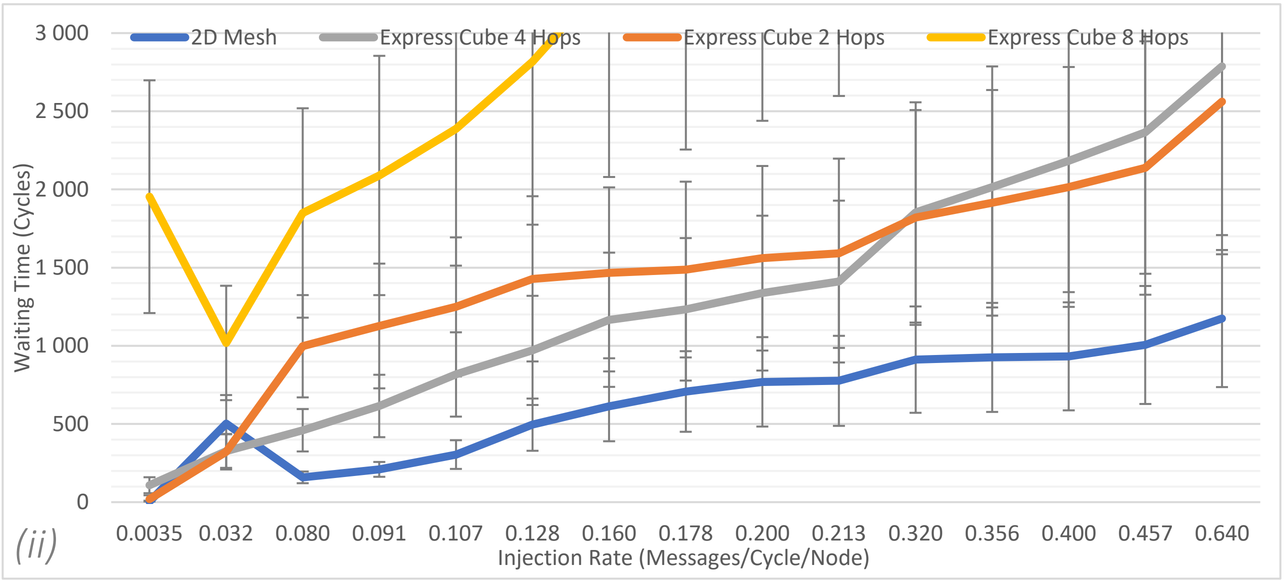


Figure 4-15 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under constrained hotspot traffic for 16x16 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput





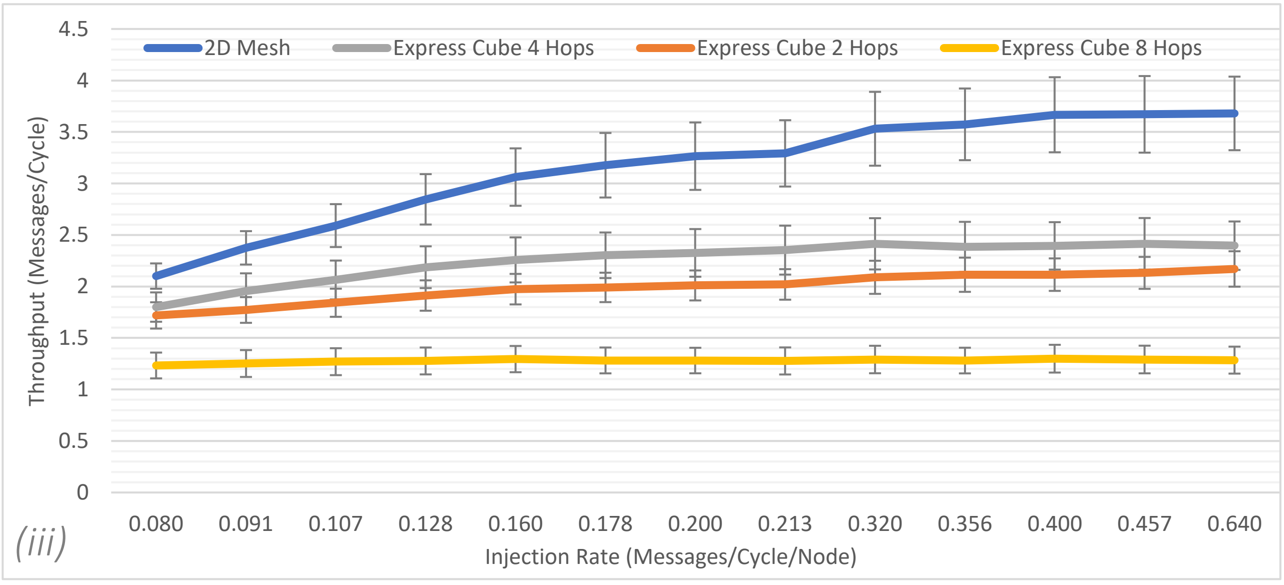


Figure 4-16 Performance results for the 2D Mesh vs EC2H vs EC4H vs EC8H under constrained hotspot traffic for 32x32 nodes (i) Response Time, (ii) Waiting Time, (iii) Throughput

4.6 Conclusions

In order to compare the performance merits of the two well-known topologies, namely the Mesh and the Express Cube, simulation results for the mean response time, the mean waiting time and throughput have been reported for a number of scenarios. For an unconstrained implementation on channel bandwidth, the Express Cube shows the best performance in terms of response time, waiting time and throughput compared to the Mesh. This is due to the combination of the express channels along with its good average distance. The EC4H exhibits performance than the EC2H and the EC8H thanks to its effective balanced bypassing of nodes. In contrast, when implementation constraints on channel bandwidth are taken into consideration the Mesh topology demonstrates superior performance over all the variations of the Express Cube topology.

This is mainly due to its higher channel bandwidth manages to offset the detrimental effects of the absence of the express channels and higher average distance in comparison to the Express Cube.

Another comparison between the Express Cube topologies has also been carried out. Simulation results have revealed that the EC4H topology deliver better performance in the unconstrained scenarios in higher node size owing to their superior graph-theoretical properties and the balanced bypassing relative to the EC2H and EC8H. However once implementation constrains are considered the significant reduction in channel bandwidth for the Express Cube topologies caused their performance to deteriorate. This allows the 2D Mesh 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 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. This is often measured in terms of the bisection width.
The aim of our project has been to compare the performance of some well-known topologies notably the Mesh and the Express Cube with its variations 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 and Git VCS version control System which is also known as source control, which is the practice of tracking and managing changes to software code. The GIT Version control systems helped us to manage changes of the simulator code source and the logic of the simulation over time. As development environments have been accelerated, version control systems help us work faster and smarter. Also, it handles all the history of the development phase and tracking and saving the reports files and traces generated from the simulation.

The link to our repository where the code source of the developed simulator is saved is:

https://github.com/ZakariaZiraoui/DiscreteEventSimulator

The repository is public, so the students and researchers working in the field of the Network of Chips can download it, use it and even modify it to match their needs and cases of studies.

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 and the Express Cube under various operating scenarios. When implementation constraints on channel bandwidth are ignored, the simulation results have indicated that the Express Cube exhibits the best performance over the Mesh topology under uniform as well as hotspot traffic patterns. This can be justified by the fact that the Express Cube has a lower average distance in comparison to the Mesh, and having a combination of the express channels that reduce long hop counts delays.

When implementation constraints on channel bandwidth are taken into consideration the 2D Mesh end up with higher channel bandwidth than the Express Cube.

The simulation results have revealed that the Mesh can take advantage of its wider channel bandwidth to mitigate the detrimental effects of the absence of the express channels and the higher average distance in comparison to the Express Cube. The latter, however, does having express links, the decrease in the bandwidth rate of its channels did not manage to exploit well and effectively those channels in order to compensate for Mesh's 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 Express Cube topologies has also been carried out. Simulation results have revealed that the EC4H topology deliver better performance in the unconstrained scenarios in higher node size owing to their superior graph-theoretical properties and the balanced bypassing relative to the EC2H and EC8H. However once implementation constrains are considered the significant reduction in channel bandwidth for the Express Cube topologies caused their performance to deteriorate. This allows the 2D Mesh 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 near future.
- Many adaptive routing algorithms have been proposed in the literature which can take advantage of the various paths that exist in a topology to improve network performance.
- A possible extension of this work would be to extend our simulator to incorporate adaptive routing and evaluate its influence on the performance properties of the NoC 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] D. Padua, Encyclopedia of parallel computing, London: Springer, 2011.

[2] K. Manna and J. Mathew, Design and Test Strategies for 2D/3D Integration for NoC-based Multicore Architectures, Springer, 2020.

[3] J. Duato, Y. Sudhakar and L. M. Ni, Interconnection Networks - An Engineering Approach, San Francisco, CA 94104-3205, USA: Elsevier Science, 2003.

[4] S. Fineberg, T. Casavant and B. Pease, "Seamless A Latency-Tolerant RISC-Based Multiprocessor Architecture," University of Iowa, December 1998.

[5] M. Ruaro, F. B. Lazzarotto, C. A. Marcon and F. G. Moraes, "DMNI: A specialized network interface for NoC-based MPSoCs," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, QC, Canada, 22-25 May 2016.

[6] K. Ray, A. Kalita, A. Biswas and M. A. Hussain, "A multipath network-on-chip topology," in *International Conference on Information Communication and Embedded Systems (ICICES)*, Chennai, India, 25-26 Feb. 2016.

[7] J. Chen, P. Gillard and C. Li, "Network-on-Chip (NoC) Topologies and Performance: A Review," in *Proceedings of the 2011 Newfoundland Electrical and Computer Engineering Conference (NECEC)*, 2011.

[8] T. N. K. Reddy, A. K. Swain, J. K. Singh and K. K. Mahapatra, "Performance assessment of different Network-on-Chip topologies," in *2nd International Conference on Devices, Circuits and Systems (ICDCS)*, Coimbatore, India, 6-8 March 2014.

[9] W. Dally, "Express cubes: improving the performance of k-ary n-cube interconnection networks," *IEEE Transactions on Computers,* vol. 40, no. 9, pp. 1016 - 1023, Sep 1991.

[10] C.-H. O. Chen, N. Agarwal, T. Krishna, K.-H. Koo, L.-S. Peh and K. Saraswat, "Comparison of Physical and Virtual Express Topologies for Future Many-core On-Chip Networks," 03 July 2014.

[11] N. E. Jerger, T. Krishna and L.-S. Peh, On-Chip Networks: Second Edition (Synthesis Lectures on Computer Architecture), Morgan & Claypool Publishers, June 19, 2017.

[12] W. J. Dally and B. P. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2004.

[13] R. Sabbaghi-Nadooshan, M. Modarressi and H. Sarbazi-Azad, "The 2D digraph-based NoCs: attractive alternatives to the 2D mesh NoCs," *The Journal of Supercomputing,* vol. 59, p. 1–21, January 2012.

[14] D. C. Jung, S. Davidson, C. Zhao, D. Richmond and M. B. Taylor, “Ruche Networks: Wire-Maximal, No-Fuss NoCs : Special Session Paper,” in *14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, Hamburg, Germany, 24-25 Sept. 2020.

[15] A.-S. Ould-Khaoua and H. Terranti, “Performance Evaluation of Networks On-chip Topologies,” University of Saad Dahleb - BLIDA 1, Blida, 2020.

[16] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling, New York NY: Wiley Computer Publishing, John Wiley & Sons, Inc, April 1991.

[17] G. Fishman, Discrete-Event Simulation Modeling, Programming, and Analysis, New York: Springer, 2001.

[18] U.S. Department of Defense, *DoD Modeling and Simulation (M&S) Verification, Validation, and Accreditation (VV&A),* 2009.

[19] O. Matoussi, “NoC Performance Model for Effcient Network Latency Estimation,” hal-03207778, Grenoble (virtual), France, Feb 2021.

[20] C.-H. O. Chen, N. Agarwal, T. Krishna, K.-H. Koo, L.-S. Peh and K. C. Saraswat, “Physical vs. Virtual Express Topologies with Low-Swing Links for Future Many-core NoCs,” in *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, Grenoble, France, 3-6 May 2010.

[21] L. P. Carloni, K. L. McMillan and A. L. Sangiovanni-Vincentelli, “Theory of Latency-Insensitive Design,” *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS,* vol. 20, no. 9, pp. 1059-1076, SEPTEMBER 2001.

[22] U. Y. Ogras and Marculescu Radu, “Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion,” in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, USA, 6-10 Nov. 2005.

[23] W. J. Dally, “Performance analysis of k-ary n-cubes interconnection networks,” *IEEE Transactions on Computers,* vol. 39, no. 06, pp. 775-785, June 1990.

[24] K. Rajeev , G. Pankaj and N. Vikas, “Network on Chip: Topologies, Routing, Implementation,” *International Journal of Advances in Science and Technology,* vol. 4, no. 1, pp. 24-34, Janury 2012.