

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Predictable Network on Chip for Real-Time Systems

Yilian Ribot González



Doctor of Philosophy on Electrical and Computer Engineering

Supervisor: Prof. Eduardo Tovar

Co-supervisor: Prof. Geoffrey Nelissen

Co-supervisor: Prof. Luis Almeida

July 23, 2024

Predictable Network on Chip for Real-Time Systems

Yilian Ribot González

Doctor of Philosophy on Electrical and Computer Engineering

Approved in oral examination by the committee:

Chair: Prof. Ana Aguiar

External Examiner: Prof. Leandro Soares Indrusiak

External Examiner: Dr. Dakshina Dasari

External Examiner: Dr. Francisco J. Cazorla

External Examiner: Prof. Pedro Souto

Co-supervisor: Prof. Geoffrey Nelissem

July 23, 2024

Abstract

Innovations in electronics have revolutionized the way we interact with the world around us. From smart homes and portable devices to autonomous vehicles and industrial automation, electronics play a pivotal role in our daily lives. At the heart of these advances is the concept of embedded systems. An embedded system is a specialized computing system designed to perform specific tasks or functions, either independently or as a part of a larger system. Real-time embedded systems are a special type of embedded system characterized by their ability to respond to events within predefined time constraints known as deadlines. Such systems are widely used in applications where timely and predictable responses are crucial, such as in automotive control systems, industrial process control, avionics, and medical devices.

These days, embedded systems are typically implemented on top of Systems-on-chip (SoCs). The growing demand for powerful embedded systems capable of performing advanced functionalities has driven a continuous rise in the number of processors embedded in SoCs. Multi-processor SoCs (MPSoCs) have become the backbone of diverse industries, finding applications in consumer electronics, aerospace, automotive, and military sectors. In these parallel architectures, Network-on-Chips (NoCs) have emerged as a compelling alternative to traditional bus-based communication architectures, effectively interconnecting all processing elements (PEs).

NoCs targeting real-time systems must ensure that packets traversing the network reach their intended destinations within defined deadlines. When it comes to developing real-time NoCs, various strategies have been explored, with three popular approaches being time-triggered routing arbitration protocol, wormhole switching with Virtual-Channels (VCs), and deflection-based routing policy. While all of these strategies have their merits, deflection-based solutions are particularly well-suited for systems with power and/or area constraints and dynamic traffic loads. This is because they typically rely on simpler routing policies and have lower design complexity when compared to NoCs based on VCs or extensive buffers. However, in deflection-based NoCs, flits can experience increased traversal times when deflecting to alternative routes after conflicting over shared resources. Consequently, it becomes crucial to reduce the number of deflections suffered by the flits or minimize the cost associated with each deflection while maintaining low hardware resource requirements, system predictability, and design simplicity. The non-deterministic number of deflections that flits may suffer while traveling through the network can lead to flits reaching their destination in a different order as they were injected at their source. To ensure in-order flit delivery, reordering mechanisms are sometimes employed at the destination nodes. However, these strategies are complex and costly, which contradicts one of the most attractive properties of deflection-based solutions: their low implementation cost.

Motivated by these topics, this thesis proposes a set of novel NoC architectures that not only address these concerns but also provide deterministic timing behaviors and flexibility to accommodate a wide variety of platforms and application requirements. Specifically, we present seven main contributions: (1) we present a buffer-less deflection-based real-time NoC with priority-based routing policy to ensure that high-priority packets receive preferential treatment over low-priority

ones when competing for network resources, thereby minimizing latency and increasing throughput for packets critical to system operation; (2) we derive a formally proven correct worst-case communication time analysis of packets with different QoS (Quality-of-Service) requirements; (3) we propose a real-time deflection-based NoC (and its timing analysis) that guarantees in-order flit delivery at the destination while avoiding the need for expensive buffers, making it suitable for resource-constrained systems; (4) we present a real-time NoC with selective flit deflection and buffering, along with its timing analysis, to prevent flits from always being deflected in contention scenarios, effectively improving their actual traversal time, preserving the in-order flit delivery feature, and maintaining the worst-case traversal time upper-bounded; (5) we propose a new strategy for regulating traffic injection based on network congestion (and a new injection time analysis) to prevent PE starvation, achieve a more balanced distribution of network bandwidth among all PEs, and avoid excessive communication delays; (6) we propose an innovative and flexible real-time higher-dimensional NoC that leverages the advantages of a fully scalable topology to enhance connectivity between PEs, potentially offering shorter paths between PEs and, consequently, more routes for parallel transmission of flits, resulting in improved communication latency; (7) and we propose a generic timing analysis for NoCs with multiple dimensions.

Keywords: Real-Time Embedded Systems, Systems-on-Chips, Network-on-Chips, Worst-Case Communication Time, Worst-Case Traversal Time, Worst-Case Injection Time, Network Congestion, Traffic Injection Regulation, Bandwidth Network Distribution

Resumo

As inovações na eletrónica revolucionaram a forma como interagimos com o mundo à nossa volta. Desde casas inteligentes e dispositivos portáteis a veículos autónomos e automação industrial, a eletrónica desempenha um papel fundamental na nossa vida quotidiana. No cerne destes avanços está o conceito de sistemas embebidos. Um sistema embebido é um sistema informático especializado concebido para executar tarefas ou funções específicas, quer de forma independente quer como parte de um sistema maior. Os sistemas embebidos em tempo real são um tipo especial de sistema embebido caracterizado pela sua capacidade de responder a eventos dentro de restrições de tempo predefinidas, conhecidas como prazos. Estes sistemas são amplamente utilizados em aplicações onde são cruciais respostas oportunas e previsíveis, como em sistemas de controlo automóvel, controlo de processos industriais, aviónica e dispositivos médicos.

Nos dias de hoje, os sistemas embebidos são tipicamente implementados em cima de Sistemas-em-chip (SoCs). A crescente procura de sistemas embebidos poderosos capazes de executar funcionalidades avançadas tem impulsionado um aumento contínuo no número de processadores incorporados nos SoCs. Os SoCs com múltiplos processadores (MPSoCs) tornaram-se a espinha dorsal de diversas indústrias, encontrando aplicações na eletrónica de consumo, aeroespacial, automóvel e militar. Nestas arquiteturas paralelas, as Redes-em-chip (NoCs) emergiram como uma alternativa atraente às arquiteturas de comunicação tradicionais baseadas em barramento, interconectando de forma eficaz todos os elementos de processamento (PEs).

As NoCs destinadas a sistemas em tempo real devem garantir que os pacotes que atravessam a rede cheguem aos seus destinos pretendidos dentro de prazos definidos. No desenvolvimento de NoCs em tempo real, várias estratégias foram exploradas, sendo três abordagens populares: protocolo de arbitragem de roteamento acionado por tempo, comutação wormhole com canais virtuais (VCs) e política de roteamento baseada em deflexão. Embora todas estas estratégias tenham os seus méritos, as soluções baseadas em deflexão são particularmente adequadas para sistemas com restrições de energia e/ou área e cargas de tráfego dinâmicas. Isto porque normalmente se baseiam em políticas de roteamento mais simples e têm menor complexidade de design quando comparadas com NoCs baseadas em VCs ou buffers extensos. No entanto, em NoCs baseadas em deflexão, os flits podem experimentar tempos de travessia aumentados quando defletem para rotas alternativas após conflito sobre recursos compartilhados. Consequentemente, torna-se crucial reduzir o número de deflexões sofridas pelos flits ou minimizar o custo associado a cada deflexão, mantendo ao mesmo tempo baixos requisitos de recursos de hardware, previsibilidade do sistema e simplicidade de design. O número não determinístico de deflexões que os flits podem sofrer enquanto viajam através da rede pode levar a que os flits cheguem ao seu destino numa ordem diferente daquela em que foram injetados na sua origem. Para garantir a entrega de flits por ordem, são por vezes empregues mecanismos de reordenação nos nós de destino. No entanto, estas estratégias são complexas e dispendiosas, o que contradiz uma das propriedades mais atraentes das soluções baseadas em deflexão: o seu baixo custo de implementação.

Motivada por estes tópicos, esta tese propõe um conjunto de arquitecturas NoC inovadoras

que não só abordam estas preocupações como também fornecem comportamentos temporais determinísticos e flexibilidade para acomodar uma vasta gama de plataformas e requisitos de aplicação. Especificamente, apresentamos sete contribuições principais: (1) apresentamos uma NoC em tempo real baseada em deflexão sem buffers com política de roteamento baseada em prioridade para garantir que os pacotes de alta prioridade recebam tratamento preferencial sobre os de baixa prioridade quando competem por recursos de rede, minimizando assim a latência e aumentando o débito para pacotes críticos para o funcionamento do sistema; (2) derivamos uma análise formalmente provada do tempo de comunicação de pior caso para pacotes com diferentes requisitos de QoS (Qualidade de Serviço); (3) propomos uma NoC em tempo real baseada em deflexão (e a sua análise temporal) que garante a entrega de flits por ordem no destino, evitando ao mesmo tempo a necessidade de buffers dispendiosos, tornando-a adequada para sistemas com restrições de recursos; (4) apresentamos uma NoC em tempo real com deflexão e buffering de flits seletivos, juntamente com a sua análise temporal, para evitar que os flits sejam sempre deflectidos em cenários de contenção, melhorando efectivamente o seu tempo de travessia real, preservando a funcionalidade de entrega de flits por ordem e mantendo o tempo de travessia de pior caso limitado superiormente; (5) propomos uma nova estratégia para regular a injeção de tráfego com base no congestionamento da rede (e uma nova análise do tempo de injeção) para prevenir que aos PE seja sempre negado acesso a um recurso compartilhado, alcançar uma distribuição mais equilibrada da largura de banda da rede entre todos os PEs e evitar atrasos de comunicação excessivos; (6) propomos uma NoC em tempo real multidimensional inovadora e flexível que aproveita as vantagens de uma topologia totalmente escalável para melhorar a conectividade entre PEs, oferecendo potencialmente caminhos mais curtos entre PEs e, consequentemente, mais rotas para a transmissão paralela de flits, resultando numa latência de comunicação melhorada; (7) e propomos uma análise temporal genérica para NoCs com múltiplas dimensões.

Palavras-chave: Sistemas Embebidos em Tempo Real, Sistemas-em-Chips, Redes-em Chip, Tempo de Comunicação de Pior Caso, Tempo de Travessia de Pior Caso, Tempo de Injeção de Pior Caso, Congestionamento de Rede, Regulação de Injeção de Tráfego, Distribuição de Largura de Banda da Rede

Acknowledgements

I would like to take this opportunity to extend my profound gratitude to all those who have played pivotal roles in my academic journey.

To my **grandma**, your recent passing has left a void in my heart that can never be filled. Your love and sacrifices were a source of strength during the happiest and most challenging moments of my life. Grandma, you have always been my guiding light, I will always miss you and forever love you. Your memory will continue to shape the person I am becoming.

To **mi Chispa**, your love, support, and understanding have been the cornerstone of this journey. As my best partner and confidant, your encouragement and belief in me have provided the strength to pursue my goals. With you, every challenge becomes an adventure, and every obstacle transforms into an opportunity. You have shown me the true meaning of love and partnership.

To my **mom** and **aunt**, for your unwavering love, encouragement, and sacrifices made throughout my whole life. You are my pillars of strength and a powerful source of motivation during this long journey, especially during the most challenging periods. Thank you for being there every step of the way. To the rest of my family, with special appreciation for my cousin **Sandra**, for your genuine concern for my career and well-wishes throughout my PhD journey.

To **Geoffrey Nelissen**, your guidance, expertise, and patience have been invaluable. Your mentorship extended beyond shaping my research skills; it also ignited a deep passion for the subject matter and instilled the confidence to conquer challenges. Your support and encouragement were constant. I am genuinely fortunate to have had the privilege of working under your guidance and wholeheartedly appreciate the knowledge you have shared.

To **Patrick Meumeu Yomsi**, who has been much more than a colleague or a friend; he has played the role of a father figure during my time in this new country. Your guidance, constant support, and caring nature have made a profound impact on my life. I am deeply grateful for your kindness and for always being there when I needed it the most. Your presence has made a huge difference.

To **Eduardo Tovar**, for providing me with the opportunity to become a member of the CISTER family. Your exceptional dedication to CISTER's success, continuous efforts to foster innovation, and ongoing support for all its members have created an outstanding work environment. Your open-door policy and mentorship have greatly enriched my academic and professional journeys.

To my friends: **Alis, Brais, Carlos, Fernando, Lily, Luis Javier, Marta, Naiza, Naylen, Reydel, Roberto, Salva, and Yaniel**, your friendship has been a source of much-needed laughter and joy, providing a comforting escape during stressful times. Thank you for your continued support and for celebrating the successes with me.

To my colleagues at CISTER: **Giann Nandi, Miguel Gutiérrez Gaitán, Jatin Arora, Ênio Filho, Jingjing Zheng, Mubarak Ojewale, Ishfaq Hussain, Yousef Emami, Radha Reddy, Harrison Kurunathan, Syed Aftab Rashid, Shashank Gaur, João Loureiro, Gowhar Javanmardi, Saeid Sabamoniri, Abdulhakeem Abdulrahman, Shardul Lendve, Alam Noor, José**

Proença, Ramiro Sámano Robles, Konstantinos Bletsas, Sérgio Penna, Pedro Santos, Kai Li, Cláudio Maia, your shared experiences and collaboration have made the academic environment a place of growth and enrichment. I am thankful for the friendships that developed within the academic community.

To the professors, especially **Luis Almeida**, your wisdom and knowledge have been instrumental in my academic and personal development.

To the administrative staff: **Cristiana Barros, Inês Almeida, Sandra Almeida, Benilde Pinto, Marwin Adorni, Henrique Quierós, David Pereira**, and **Filipe Pacheco**, whose support and assistance have been invaluable in making my academic journey much easier.

I express my gratitude to **Fernando** and **Giann** for their translation of the abstract into Portuguese. Muito Obrigada!

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); by national funds, through the FCT/MCTES under the scope of the project LASI-LA/P/0104/2020 (LASI); by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT); and by FCT and the ESF (European Social Fund) through the Regional Operational Programme (ROP) Norte 2020, under PhD grant 2020.06898.BD.

This thesis represents a culmination of our collective efforts. Thank you for being an integral part of my success.

Yilian Ribot González

Publications

Outputs of this research:

1. **Y. Ribot González**, G. Nelissen, and E. Tovar. Traffic Injection Regulation Protocol based on free time-slots requests. In 29th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2023.
2. **Y. Ribot González**, G. Nelissen, and E. Tovar. IPDeN 2.0: Real-time NoC with selective flit deflection and buffering. In Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS), pp. 87–98, 2023, doi:10.1145/3575757.3575758.
3. **Y. Ribot González**, G. Nelissen and E. Tovar, IPDeN: Real-Time deflection-based NoC with in-order flits delivery. In 28th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 160-169, 2022, doi:10.1109/RTCSA55878.2022.00023.
4. **Y. Ribot González**, G. Nelissen, and E. Tovar. nDimNoC: Real-Time D-dimensional NoC. In 33rd Euromicro Conference on Real-Time Systems (ECRTS), vol. 196, pp. 5:1-5:22, 2021, doi:10.4230/LIPIcs.ECRTS.2021.5
5. **Y. Ribot González** and G. Nelissen. HopliteRT*: Real-Time NoC for FPGA. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3650-3661, 2020, doi:10.1109/TCAD.2020.3012748.
6. **Y. Ribot González**, and G. Nelissen. Design and implementation of an FPGA-based NoC for Real Time Systems. In Proceedings of the Work-in-Progress Session (ECRTS): 1-3, 2019.

Other publications:

1. J. Pérez Rodríguez, P. Meumeu Yomsi, **Y. Ribot González**, and L. J. Pente Lam. B-TSP: An Advanced Power Safe Management Strategy for modern Multi-core Platforms under Thermal-Aware Design. In Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS), pp. 34–44, 2023, doi:10.1145/3575757.3593659.

"A goal without a deadline is just a dream."

Robert Herjavec

Contents

1	Introduction	1
1.1	Real-Time Embedded System	1
1.2	Multi-Processor Architecture for Embedded Systems	2
1.2.1	Intercommunication Medium	2
1.2.1.1	Shared Communication Busses	2
1.2.1.2	Network-on-Chip Paradigm	3
1.3	Thesis Statement	4
1.4	Specific Objectives	5
1.5	Overview of the Contributions	5
1.6	Thesis Structure	7
2	Background and Related Work	9
2.1	NoC Architectures	9
2.1.1	Network Topology	9
2.1.2	Routing Mechanism	11
2.1.3	Switching Mechanism	12
2.1.4	Flow-Control Strategy	13
2.1.5	Contention scenarios	15
2.1.6	Arbitration Mechanisms	16
2.2	NoCs for Real-Time Systems	17
2.2.1	NoCs Based on Wormhole Switching with VCs	17
2.2.1.1	Worst-Case Network Latency Analysis for Wormhole-Based NoCs	19
2.2.2	NoCs with a Deflection-Based Routing Policy	20
2.2.3	Other NoCs architectures	20
2.3	HopliteRT NoC: Overview	22
2.3.0.1	System Model	22
2.3.0.2	Network Topology and Router Architecture	22
2.3.0.3	Routing Policy	22
2.3.0.4	Traffic Injection Regulation Protocol	24
2.3.0.5	Worst-case Communication Time Analysis	25
2.4	FPGA Platform in the Scope of NoCs	26
2.5	Chapter Summary	26
3	Revisiting HopliteRT for Improved Real-Time Performances	27
3.1	System Model	29
3.2	HopliteRT* Design	29
3.2.1	Priority-Based Routing Policy	29
3.2.2	Ensuring Progress: Circulant Topology on Time Determinism	31

3.2.3	Router Architecture	33
3.3	Discussion	35
3.3.1	Traffic Injection Discussion	35
3.3.2	Livelock and Deadlock Situations	36
3.4	Worst-case Communication Time Analysis	36
3.4.1	Bound on the Worst-Case Traversal Time	37
3.4.2	Improved Analysis for the Worst-Case Traversal Time	39
3.4.3	Bound on the Worst-Case Injection Time	43
3.5	Experimental Results	47
3.5.1	Implementation of HopliteRT*	47
3.5.2	NoC Latency Bounds for randomly generated flows	48
3.5.3	Use-case analysis	51
3.5.4	RTL Simulations	51
3.6	Chapter Summary	53
4	Ensuring In-Order Flit Delivery at the Destination	55
4.1	System Model	56
4.2	IPDeN Design	57
4.2.1	Routing Policy	57
4.2.2	Router Architecture	58
4.3	Discussion	59
4.3.1	Buffer Size Analysis	59
4.3.2	Livelock and Deadlock Situations	61
4.3.3	Priority Levels	61
4.4	Worst-Case Communication Time Analysis	61
4.5	Network Latency Discussion	63
4.6	Experimental Results	63
4.6.1	Implementation of IPDeN	63
4.6.2	RTL Simulations	64
4.6.3	Test Cases	65
4.7	Chapter Summary	73
5	Improving Average Worst-Case Traversal Time	75
5.1	System Model	77
5.2	IPDeN 2.0 Design	77
5.2.1	Router Architecture	77
5.2.2	Routing Policy	78
5.2.3	In-order flit delivery	83
5.3	Worst-Case Communication Time Analysis	84
5.4	Experimental Results	85
5.4.1	Implementation	85
5.4.2	Simulation Results	86
5.5	Chapter Summary	95
6	Traffic Injection Regulation Protocol	97
6.1	Related Work	98
6.2	System Model	99
6.3	Our Solution	99
6.3.1	Implementation	105

6.3.2	Discussion	106
6.4	Worst-Case Injection Time Analysis	106
6.5	Experimental Results	109
6.5.1	Synthetic Test Cases	109
6.5.2	Orion Test Case	115
6.6	Chapter Summary	119
7	Attempt to Generalize to a Higher-Dimensional NoC Architecture	121
7.1	Related Work	122
7.2	System Model	122
7.3	NoC Design	123
7.3.1	Network Topology	123
7.3.2	Router Architecture	124
7.3.3	Routing Policy	126
7.4	Worst-Case Communication Time Analysis	128
7.4.1	Best- and Worst-Case Traversal Time Analysis	128
7.4.2	Bound on the Worst-Case Injection Time	135
7.5	Experimental Results	138
7.5.1	Implementation	138
7.5.2	NoC Latency Bounds	139
7.6	Chapter Summary	141
8	Thesis Summary, Conclusion and Future Work	143
8.1	Summary of Contributions	143
8.2	Thesis Validation	144
8.3	Future work	146
8.3.1	Fault Tolerance	146
8.3.2	Energy Efficiency	147
8.3.3	Applications and Workload Scenarios	147
A	Counter-Example to the WCIT Bound of [90, 91]	149
B	Experimental Results for IPDeN NoC	151
C	Experimental Results for IPDeN 2.0 NoC	167
	References	177

List of Figures

1.1	Bus-based communication architecture.	3
1.2	Conceptual view of a NoC and packet transfer over NoC.	3
2.1	Network Topologies.	10
2.2	Examples of X-Y routing.	12
2.3	Example of Store-and-Forward technique.	13
2.4	Example of Wormhole Switching technique.	13
2.5	Conceptual view of credit-based flow control strategy.	14
2.6	Virtual Channels located in an input port i	15
2.7	Example of contention scenarios.	15
2.8	Example of deflection-based routing policy.	16
2.9	Example of routers based on different strategies.	18
2.10	HopliteRT design.	23
2.11	Example of HopliteRT's routing policy.	23
2.12	Example of a contention scenario in HopliteRT.	24
3.1	Example of the penalties for deflecting flits in HopliteRT.	28
3.2	Example of HopliteRT*'s priority-based routing policy.	30
3.2	Example of HopliteRT*'s priority-based routing policy (cont.).	31
3.3	HopliteRT* network topology.	33
3.4	HopliteRT* router architecture.	34
3.5	Example of traffic injection in HopliteRT* NoC.	35
3.5	Example of traffic injection in HopliteRT* NoC (cont.).	36
3.6	Example of how to calculate an upper-bound on the maximum number of deflections suffered by a flit of a high-priority flow.	43
3.7	Experimental results for 16x16 NoC.	49
3.8	Experimental results for 4x4 NoC and random traffic pattern.	50
3.9	Experimental results for the case study.	51
3.10	Experimental results for 4x4 NoC and random traffic pattern.	52
3.11	WCTT of randomly generated packets traversing a 4x4 NoC.	53
4.1	Example of an out-of-order reception in HopliteRT*.	56
4.2	IPDeN router architecture.	58
4.3	Example of how buffers are used in IPDeN.	60
4.4	Average measured communication time for random traffic.	65
4.5	Experimental results for Synthetic and Orion test case (traversal time).	69
4.5	Experimental results for Synthetic and Orion test case (traversal time) (cont.). . .	70
4.6	Experimental results for Synthetic and Orion test case (communication time). . .	71
4.6	Experimental results for Synthetic and Orion test case (communication time) (cont.).	72

5.1	Example of the penalties for deflecting and buffering flits in IPDeN.	76
5.2	Architecture of the routers in IPDeN and IPDeN 2.0 NoCs.	77
5.3	Example of IPDeN's routing policy.	79
5.4	Example of IPDeN 2.0's routing policy.	80
5.5	Example of in-order flits delivery property in IPDeN 2.0.	83
5.5	Example of in-order flits delivery property in IPDeN 2.0 (cont.).	84
5.6	IPDeN 2.0 vs IPDeN (Orion test case).	87
5.6	IPDeN 2.0 vs IPDeN (Orion test case) (cont.).	88
5.7	IPDeN 2.0 vs 128-deep HopliteBuf (Orion test case).	90
5.7	IPDeN 2.0 vs 128-deep HopliteBuf (Orion test case) (cont.).	91
5.8	IPDeN 2.0 vs VCs (Orion test case).	92
5.8	IPDeN 2.0 vs VCs (Orion test case) (cont.).	93
6.1	Example 1 of D-TSR protocol in HopliteRT* NoC.	101
6.1	Example 1 of D-TSR protocol in HopliteRT* NoC (cont.).	102
6.2	Example 2 of D-TSR protocol in HopliteRT* NoC.	102
6.2	Example 2 of D-TSR protocol in HopliteRT* NoC (cont.).	103
6.3	Example 3 of D-TSR protocol in HopliteRT* NoC.	104
6.3	Example 3 of D-TSR protocol in HopliteRT* NoC (cont.).	105
6.4	Inputs, outputs, and signals in HopliteRT* + D-TSR.	105
6.5	Test cases.	109
6.6	Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 100.	110
6.6	Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 100 (cont.).	111
6.7	Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 20.	112
6.7	Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 20 (cont.).	113
6.8	Number of injected flits in given time intervals when PEs are on the same column of the network and have utilization equal to 100.	114
6.8	Number of injected flits in given time intervals when PEs are on the same column of the network and have utilization equal to 100 (cont.).	115
6.9	Number of injected flits in given time intervals under all-to-one traffic pattern when all PEs have utilization equal to 100.	116
6.9	Number of injected flits in given time intervals under all-to-one traffic pattern when all PEs have utilization equal to 100 (cont.).	117
6.10	Number of injected flits for the Orion test case.	118
6.10	Number of injected flits for the Orion test case (cont.).	118
7.1	nDimNoc network topology.	123
7.2	nDimNoc router architecture.	125
7.3	Example of nDimNoc's routing policy.	128
7.4	DAG of the potential trajectories that a flit may take from the origin (0;0;1) to the destination (3;1;0) when it traverses a 4x2x2 3-D nDimNoC.	130
7.5	Experimental results for a random traffic pattern.	140
A.1	Counter-example to the WCIT bound of [90, 91]	149

B.1	Experimental results for Synthetic test case.	151
B.1	Experimental results for Synthetic test case (cont. 1).	152
B.1	Experimental results for Synthetic test case (cont. 2).	153
B.2	Experimental results for Orion test case (WMTT).	153
B.2	Experimental results for Orion test case (WMTT) (cont. 1).	154
B.2	Experimental results for Orion test case (WMTT) (cont. 2).	155
B.2	Experimental results for Orion test case (WMTT) (cont. 3).	156
B.3	Experimental results for Orion test case (WMCT).	156
B.3	Experimental results for Orion test case (WMCT) (cont. 1).	157
B.3	Experimental results for Orion test case (WMCT) (cont. 2).	158
B.3	Experimental results for Orion test case (WMCT) (cont. 3).	159
B.4	Experimental results for Orion test case (AMTT).	160
B.4	Experimental results for Orion test case (AMTT) (cont.1).	161
B.4	Experimental results for Orion test case (AMTT) (cont.2).	162
B.4	Experimental results for Orion test case (AMTT) (cont. 3).	163
B.5	Experimental results for Orion test case (AMCT).	163
B.5	Experimental results for Orion test case (AMCT) (cont. 1).	164
B.5	Experimental results for Orion test case (AMCT) (cont. 2).	165
B.5	Experimental results for Orion test case (AMCT) (cont. 3).	166
C.1	IPDeN 2.0 vs IPDeN (Synthetic test case).	167
C.1	IPDeN 2.0 vs IPDeN (Synthetic test case) (cont.).	168
C.2	IPDeN 2.0 vs 128-deep HopliteBuf (Synthetic test case).	168
C.2	IPDeN 2.0 vs 128-deep HopliteBuf (Synthetic test case) (cont.).	169
C.3	IPDeN 2.0 vs VCs (Synthetic test case) (cont.).	170
C.3	IPDeN 2.0 vs VCs (Synthetic test case) (cont.).	171

List of Tables

2.1	Routing Table of HopliteRT	24
3.1	Routing Table of HopliteRT*	32
3.2	Resources utilization of different NoCs in Kintex-7 FPGAs.	48
4.1	Routing Table of IPDeN.	58
4.2	Resources utilization of one router.	64
4.3	Configuration details for the test cases.	66
4.4	IPDeN vs HopliteBuf (Synthetic test case).	67
4.5	IPDeN vs HopliteBuf (Orion test case).	67
4.6	IPDeN vs VCs-based NoC (Synthetic test case).	67
4.7	IPDeN vs VCs-based NoC (Orion test case).	68
4.8	IPDeN vs HopliteBuf and IPDeN vs VCs-based NoC (Synthetic test case).	68
4.9	IPDeN vs HopliteBuf and IPDeN vs VCs-based NoC (Orion test case).	68
5.1	Routing Table of IPDeN 2.0.	82
5.2	Resources utilization of different NoCs in Kintex-7 FPGAs.	86
5.3	Configuration details for the test cases.	87
5.4	IPDeN 2.0 vs IPDeN (Orion test case).	88
5.5	IPDeN 2.0 vs 5-deep HopliteBuf (Orion test case).	89
5.6	IPDeN 2.0 vs 128-deep HopliteBuf (Orion test case).	91
5.7	Number of lost flits (Orion test case).	91
5.8	Analysis of the lost flits in HopliteBuf (Orion).	91
5.9	IPDeN 2.0 vs VCs-based NoC (Orion test case).	93
5.10	IPDeN 2.0 vs IPDeN (Synthetic test case).	94
5.11	IPDeN 2.0 vs 128-deep HopliteBuf (Synthetic test case).	94
5.12	IPDeN 2.0 vs VCs-based NoC (Synthetic test case).	94
6.1	Configuration details for Orion test case.	116
7.1	Generic routing policy table of nDimNoC with D dimensions	127
7.2	Resources utilization of different NoCs in Kirtex-7 FPGAs.	138
C.1	Gain on the WMTT (Orion test case).	171
C.2	Gain on the WMCT (Orion test case).	172
C.3	Gain on the AMTT (Orion test case).	172
C.4	Gain on the AMCT (Orion test case).	173
C.5	Lost on the WMTT (Orion test case).	173
C.6	Lost on the WMCT (Orion test case).	174

C.7	Lost on the AMTT (Orion test case).	174
C.8	Lost on the AMCT (Orion test case).	175

Abbreviations

2D-NoC	2-Dimensional NoCs
3D-IC	3-Dimensional Integrated Circuit
ASIC	Application-Specific Integrated Circuits
ATM	Automatic teller machines
BCCT	Best-Case Communication Time
DAG	Directed Acyclic Graph
EDF	Earliest Deadline First
FPGA	Field-Programmable Gate Array
GPS	Global Positioning Systems
HDL	Hardware Description Language
IC	Integrated Circuit
IP	Intellectual Property
MPSoC	Multi-Processor Systems-on-chip
NI	Network Interface
NoC	Network-on-Chip
PE	Processing Element
QoS	Quality-of-Service
SoC	Systems-on-chip
SotA	State-of-the-Art
WCCT	Worst-Case Communication Time
WCIT	Worst-Injection Time
WCTT	Worst-Case Traversal Time

Chapter 1

Introduction

1.1 Real-Time Embedded System

Everywhere in our daily lives, we are confronted to electronic devices that provide specific and particular services. Such devices are called *embedded systems*. Examples of these devices include:

- ▷ Automatic Teller Machines (ATMs): These machines enable individuals to perform multiple banking transactions, including cash withdrawals, deposits, balance inquiries, and fund transfers.
- ▷ GPS Systems (Global Positioning Systems): Offer accurate location and navigation information by leveraging satellite signals, assisting in mapping, providing directions, and tracking locations.
- ▷ Gaming Consoles: Specially designed for playing video games, offering immersive gaming experiences with advanced graphics and interactive gameplay.
- ▷ Industrial Robots: Found in manufacturing and industrial settings, industrial robots are automated machines used to perform tasks like assembly, welding, painting, and material handling, enhancing efficiency and precision.
- ▷ Aircraft Control System: This system is responsible for maintaining the stability, maneuverability, and safety of the aircraft during all phases of flight, including takeoff, landing, and cruising.

These systems are subject to tight design constraints in terms of cost, size, performance, and power consumption. To meet their various design requirements, embedded systems rely on various processing hardware such as microcontrollers, microprocessors, Field-Programmable Gate Array (FPGAs), and Application-Specific Integrated Circuits (ASICs).

Depending on the purpose, embedded systems can be *non-real-time* or *real-time*. *Non-real-time embedded systems* are those systems that operate without strict timing constraints, with their

primary focus on executing their functions accurately. They may prioritize factors such as efficiency, cost-effectiveness, and power consumption. Some examples include but are not limited to, a blood pressure meter, whose primary function is to calculate the blood pressure based on sensing data, or an automated assembly chain that must perform specific operations in a predetermined sequence. However, there are a variety of applications where events or requests must be treated according to predefined timing requirements, e.g., the inflation of an airbag in the event of a head-on collision to reduce the risk of the driver hitting the interior of the vehicle. Systems with timing requirements associated to their tasks are called *real-time systems*, and their correctness depends on both the logical results of the computation and their temporal behaviour [83]. Applications of real-time systems include, but are not limited to: engine control systems, flight control systems, satellite communications, medical devices for patient monitoring, controllers for automated building and industrial processes, power grid controllers, and missile guidance systems.

In real-time embedded systems such as the airbag control system in cars, it is crucial that all tasks are completed within the predefined deadlines. If a deadline is missed, the result is useless, even if it is correct. Missing important deadlines can lead to serious consequences, including death, damage to the infrastructure or environment, or loss of money. Such systems are referred to as *hard real-time embedded systems*. Conversely, in some other devices, performance, and therefore quality of service, are not completely compromised when timing constraints are not met. This is the case, for instance, in a video game, where an occasional lag (i.e., the time delay between a gamer's action and the game engine's response to that input) may degrade the gaming experience but does not render the system useless or dangerous. Those systems are still considered to be real-time but are referred to as soft real-time systems.

1.2 Multi-Processor Architecture for Embedded Systems

Embedded systems are typically implemented on top of Systems-on-chip (SoCs) that can vary in complexity and composition. The simplest SoCs consist of a single processor. However, most SoCs consist of multiple, possibly heterogeneous, processing elements that execute multiple processes in parallel. In fact, Multi-processor SoCs (MPSoCs) are the backbone of mobile devices, automotive systems, high-performance computing, aerospace and defense applications, networking and communication, medical devices, and more [2].

1.2.1 Intercommunication Medium

1.2.1.1 Shared Communication Busses

In MPSoCs, the conventional approach for communication among processing elements (PEs), memory, and I/O devices relies on shared buses. This simplistic topology was widely adopted due to its ease of extension and low cost in terms of silicon area (see Figure 1.1). However, with the rising demand for more powerful embedded systems capable of performing advanced functions, the number of PEs integrated into SoCs has increased significantly in the last decade. Consequently,

the utilization of shared communication buses in these platforms has become inadequate. This is primarily attributed to their limited bandwidth and scalability when it comes to accommodating the increasing number of message exchanges between PEs.

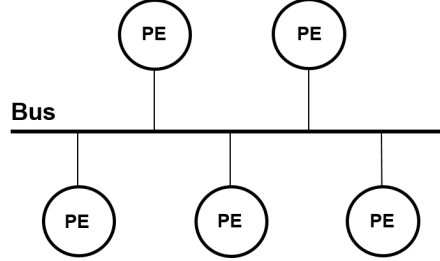


Figure 1.1: Bus-based communication architecture.

Shared buses exhibit a fundamental limitation: only one node can control the bus and transmit data at any given clock cycle. As a result, this traditional communication mechanism has been identified as a significant bottleneck that negatively affects the performance of SoCs. Therefore, it has become crucial to explore alternative communication architectures that can effectively tackle the challenges arising from the growing number of PEs.

1.2.1.2 Network-on-Chip Paradigm

Network-on-chips (NoCs) is a new communication infrastructure that emerged as a suitable alternative to the more traditional bus-based communications architectures [7, 38]. Nowadays, they have been widely adopted in modern MPSoCs, as evidenced in [92, 43, 87, 44, 48, 20].

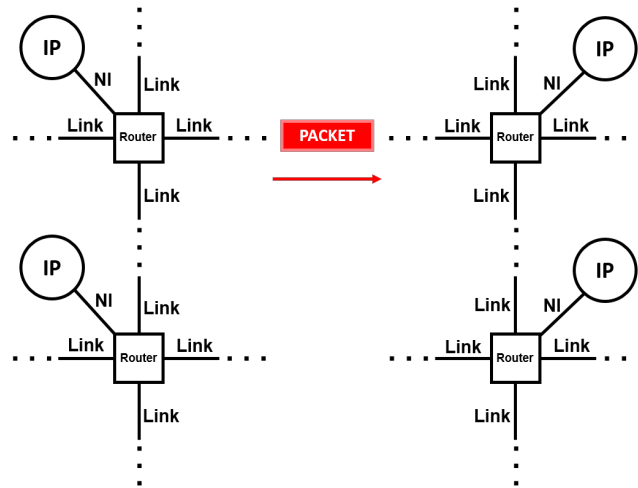


Figure 1.2: Conceptual view of a NoC and packet transfer over NoC.

NoCs are router-based packet-switching networks, where data is split into small packets and transmitted through the network. Specifically, they consist of network interfaces, routers, and links (see Figure 1.2). IP (Intellectual Property)-blocks such as processors and memories, are connected

to routers via network interfaces (NIs). Routers then route packets from a source IP to a destination IP following a communication protocol. The routers in the network are physically interconnected by links, which enable data transmission between routers.

Parallelism plays a crucial role in NoC architectures by facilitating concurrent communication among multiple PEs. Unlike traditional shared bus architectures, NoCs utilize a network of routers and links to establish multiple communication paths. This parallel architecture enables simultaneous data transfers, effectively mitigating communication bottlenecks and enhancing overall system performance.

1.3 Thesis Statement

NoCs have remarkable advantages in terms of scalability, parallelism, and reusability properties [7, 38]. However, the use of NoCs in real-time systems introduces an additional layer of complexity. NoCs for real-time systems must provide timing guarantees, i.e., packets traversing the network must reach their destination within previously defined deadlines [39]. Ideally, real-time NoCs must provide guaranteed service for critical tasks by meeting their timing constraints while providing an acceptable average response time for non-critical tasks. However, NoCs have much more complex architectures and communication protocols than traditional communication buses. Namely, NoCs consist of numerous shared resources (e.g., routers, links) that can be subject to contention scenarios when multiple communication flows compete for access to those resources. As a result, NoCs are difficult to analyze from a timing perspective, and it is hence difficult to provide guarantees on the communication time between different PEs. Designing a predictable NoC architecture is therefore an important challenge for systems with timing constraints.

There are several NoC architectures in the literature that target real-time systems. We discuss some of them in Chapter 2. However, NoCs that ensure deterministic behavior are often restricted to static/non-evolving applications or are unfit for systems with small/medium production yield and/or cost or power consumption constraints. From a real-time analysis perspective, upper bounds have been proposed on the worst-case communication time (WCCT) of packets traversing such NoC solutions, e.g., [80, 82, 23, 88, 18, 74, 53, 62, 13, 42]. However, the complexity of the NoC designs and their routing policies make their timing analysis complex, resulting in a range of challenges and issues.

In this context, NoCs built upon the concept of deflection are a good alternative, as they have lower implementation cost and power consumption compared to other designs [58, 49, 90, 77, 78, 25]. Additionally, they typically rely on simpler routing policies that are easier to analyze. Nonetheless, most of the existing deflection-based solutions that provide real-time guarantees favor average over worst-case timing performance. Indeed, until very recently, there have been only a few real-time deflection-based NoCs that exhibit predictable timing behavior and for which a complete analysis of the worst-case communication time of packets exists, e.g., [90, 25].

Motivated by these considerations, the thesis statement supported by this dissertation is the following.

It is possible to design and implement a predictable deflection-based NoC architecture for real-time systems that not only provides different levels of QoS and offers deterministic and easy-to-analyze timing behaviors but also guarantees in-order flit delivery at the destination and ensures efficient network bandwidth utilization, while remaining suitable for implementation in both low-end and high-end chips, and for which a comprehensive analysis of the worst-case communication time of packets can be provided.

1.4 Specific Objectives

The limitations of existing NoC solutions discussed so far highlight the challenges in providing strong real-time guarantees for systems implemented upon MPSoCs. To address these limitations, we will design alternative NoC architectures specifically tailored for a wide range of industrial applications. The developed NoC architectures will ensure predictable and easy-to-analyze timing behaviors, ease of configuration, configurable Quality-of-Service (QoS), and communication throughput to traffic flows. They will also be amenable to dynamic workloads, and scale to systems with an extensive number of processing elements and/or communicating tasks while guaranteeing high performance. Additionally, these architectures will be suitable for most systems, including those with low power consumption and/or limited resource availability. More specifically, our objectives include:

- ▷ Design and implement NoC architectures built upon the notion of deflection to reduce the reliance on extensive buffers and VCs in the real-time communication infrastructure of the next generation of MPSoCs. These NoC architectures will support a priority-based routing policy and ensure in-order flit delivery at the destination.
- ▷ Develop a formally proven correct timing analysis for each of the proposed NoC architectures.
- ▷ Implement all the proposed NoC solutions in a hardware description language (HDL) to enable integration into future SoCs. This approach ensures that the developed NoCs can be utilized and deployed in real systems.

1.5 Overview of the Contributions

HopliteRT is a deflection-based NoC proposed a few years ago in [90] that caught our attention because, among all the existing NoC architectures in the literature, it closely aligns with our research objectives. Specifically, HopliteRT is a buffer-less design that ensures predictable and easy-to-analyze timing behavior, is amenable to dynamic workloads, and has low implementation costs. However, it does not provide different QoS to communication flows and nowadays this is a requirement of many real-time embedded systems. Moreover, it does not guarantee that flits arrive at the destination in the same order as they were injected at their sources. Ensuring in-order packet

delivery is crucial for a wide range of applications such as control systems where actions depend on specific input sequences, or applications transmitting large data blocks that require complete data reconstruction at the destination. Finally, in the worst case, flits may traverse multiple routers of HopliteRT NoC twice, leading to an increased number of hops from origin to destination, and therefore longer communication times of packets.

HopliteRT served as the starting point for this research. Our initial contribution aimed to *enhance the worst-case traversal time (WCTT) of packets* by introducing new properties and modifying existing ones in the HopliteRT architecture. Specifically, we changed the network topology and introduced the notion of packet priority into the design. The core idea behind this approach is to prioritize high-priority packets over low-priority ones when competing for the same network resources. This involves assigning shorter paths to high-priority packets and granting them access to the network before low-priority ones. As a result, the latency of critical packets essential to system operation decreases. For the resulting NoC architecture, we *derived a comprehensive analysis of the worst-case communication time (WCCT) of packets*, which is computed by the worst-case traversal time (WCTT) and the worst-case injection time (WCIT). The non-deterministic number of deflections suffered by the flits while traversing the network leads to potential routing of flits from the same packet through different paths with different lengths, causing out-of-order flits delivery at their destinations. Therefore, we *introduced the in-order flit delivery property* into the design without relying on complex and costly reordering mechanisms at the destination nodes, which is a common practice in many existing works. However, in the resulting solution, flits may be deflected multiple times to alternative routes while traversing the network toward their destination, leading to potentially large traversal times. Therefore, we yet *enhanced the design of our NoC to improve the average traversal time of packets*. The timing analysis was also extended to accommodate the new network properties and behavior. Finally, to avoid PEs starvation due to flits traveling along the network and blocking the injection of new flits, we *designed a new protocol to regulate traffic injection and manage QoS of communication flows*. All the intermediate NoC solutions resulting from each stage were implemented in a hardware description language and synthesized on an FPGA platform. The solution resulting from the integration of all the contributions mentioned above is a predictable deflection-based NoC that provides configurable QoS to traffic flows, ensures in-order flit delivery at the destination, includes a traffic injection regulation protocol, guarantees bounded WCCT, and ensures efficient utilization of hardware resources, which meets our initial goal.

As an additional or bonus contribution going further than our original objective, we *leveraged the higher-dimensional property of the network topology* used in our NoC architecture to *increase the number of router interconnections*. This allows for the parallel transmission of a greater number of flits through the network, potentially decreasing the traversal time of some flits due to the availability of more, and possibly shorter, routes. Furthermore, this enhanced network architecture improves the injection time of packets, since with more links between routers, PEs have access to additional output ports for injecting their flits. This research direction is further motivated by the recent advancements in the integrated circuit (IC) industry. The future of ICs

appears to lie in three-dimensional integrated circuits (3D-ICs), which offer numerous advantages such as improved performance, reduced interconnection length, higher packing density due to the additional dimension, reduced power consumption, and enhanced computation bandwidth. As a result, there is a growing push to develop robust NoC solutions that effectively address the needs of future large-scale MPSoCs by leveraging the combined benefits of 3D integration and NoC architecture. For this resulting architecture, we derived a WCCT analysis, specifically employing Directed Acyclic Graphs (DAGs) to compute packet's WCCT.

1.6 Thesis Structure

The remainder of this dissertation is structured as follows:

- ▷ Chapter 2 is the cornerstone for the subsequent chapters, providing a comprehensive overview of the current State-of-the-Art (SotA) in real-time NoCs. Its primary objective is to establish a solid foundation for a deeper understanding of the advancements and challenges faced in designing efficient and reliable NoC architectures for real-time systems. It offers a detailed background on essential NoC design features and analyzes various solutions from the literature, each relying on different philosophies and approaches.
- ▷ Chapter 3 marks the first step towards achieving the goals of this research. It provides a predictable and cost-efficient NoC architecture for MPSoC that meets the requirements of real-time systems and supports a priority-based routing policy. This policy is built upon the notion of deflection, leveraging the properties of a fully scalable topology. The chapter thoroughly explores the NoC's design features, including its routing policy, network topology, and router architecture. Additionally, it offers a detailed analysis of the WCCT of packets with different priorities traveling along the network. Finally, it presents various experimental results demonstrating the capabilities and advantages of this innovative NoC design.
- ▷ Chapter 4 extends the previous solution to support in-order flit delivery at the destination by adding a small and fixed-size buffer to each router. In addition, it extends the timing analysis derived in Chapter 3 to accommodate these new network features, specifically addressing the impact of buffers on packet communication delays. Finally, it compares the resulting architecture against the SotA to demonstrate its advantages and improvements.
- ▷ Chapter 5 provides an enhanced version of the previous architecture that combines the benefits of small buffers with the concept of deflection to prevent flits from always being deflected in contention scenarios, thereby improving the average packet traversal time. Correspondingly, it also updates the timing analysis and evaluates this improved version against its predecessor and other NoCs solutions.
- ▷ Chapter 6 provides a novel strategy for regulating traffic injection based on network congestion, which limits the network bandwidth that PEs and/or communication flows can claim.

Additionally, it derives a new upper-bound on the WCIT of flits when traffic injection is regulated by this protocol. Finally, it discusses the results of multiple sets of experiments conducted to evaluate the effectiveness of this approach.

- ▷ Chapter 7 provides a new buffer-less and flexible router architecture that allows synthesizing NoCs with multiple dimensions and bounded worst-case communication delays, aimed at reducing average network communication latency and offering greater flexibility compared to traditional 2D-NoCs. Additionally, it develops a WCCT analysis where the WCTT of packets is computed using DAG. Finally, it compares the solution against relevant works.
- ▷ Chapter 8 offers an overview of the conducted research, summarizing the key findings and results, discussing any open questions that emerged during the research process, and providing suggestions for potential future research directions.

Chapter 2

Background and Related Work

Over the years, various approaches have been explored in designing real-time NoCs. Section 2.1 delves into the design features of NoCs, while Section 2.2 offers an overview of existing literature on NoCs for real-time systems. Sections 2.2.1 and 2.3 look into specific NoC solutions that are based on the two prevalent approaches for real-time NoC design. This analysis allows us to evaluate the strengths and weaknesses of each approach. Finally, Section 2.4 discusses the role of FPGA platforms in the context of NoCs.

2.1 NoC Architectures

Designing a NoC is a complex process that requires consideration of a variety of factors, including the number and type of IP-blocks, communication patterns, bandwidth requirements, chip power consumption and area constraints, and the desired level of fault tolerance. To overcome these challenges, NoC designers combine various design aspects, e.g., network topology, routing mechanism, switching mechanism, flow-control strategies, arbitration mechanisms, and the specific type of hardware components used for their implementation.

2.1.1 Network Topology

In NoCs, network topology refers to the arrangement of IP blocks, NIs, routers, and physical links on the chip and how they are interconnected. The topology defines the different paths that packets can take to reach their destination, and thus the number of routers they reach when traversing the network along one of these paths. It also determines the length of the links between each pair of routers. Therefore, the topology plays a crucial role in the latency, energy consumption, and bandwidth of the network

The most commonly used network topologies in NoCs include *ring* (see Figure 2.1a), *star* (see Figure 2.1b), *mesh* (see Figure 2.1c), *torus* (see Figure 2.1d), *tree* (see Figure 2.1e) or *butterfly* (see Figure 2.1f). In a mesh topology (see Figure 2.1c), for example, all nodes (where each node is defined by a router and its corresponding connected PE) are connected in a grid pattern. Each node then has 2 to 4 neighboring nodes, depending on its position in the grid. The torus topology can

be achieved by adding direct links between the first and the last node of the same row or column to the mesh architecture (see Figure 2.1d). All nodes have four neighbors then.

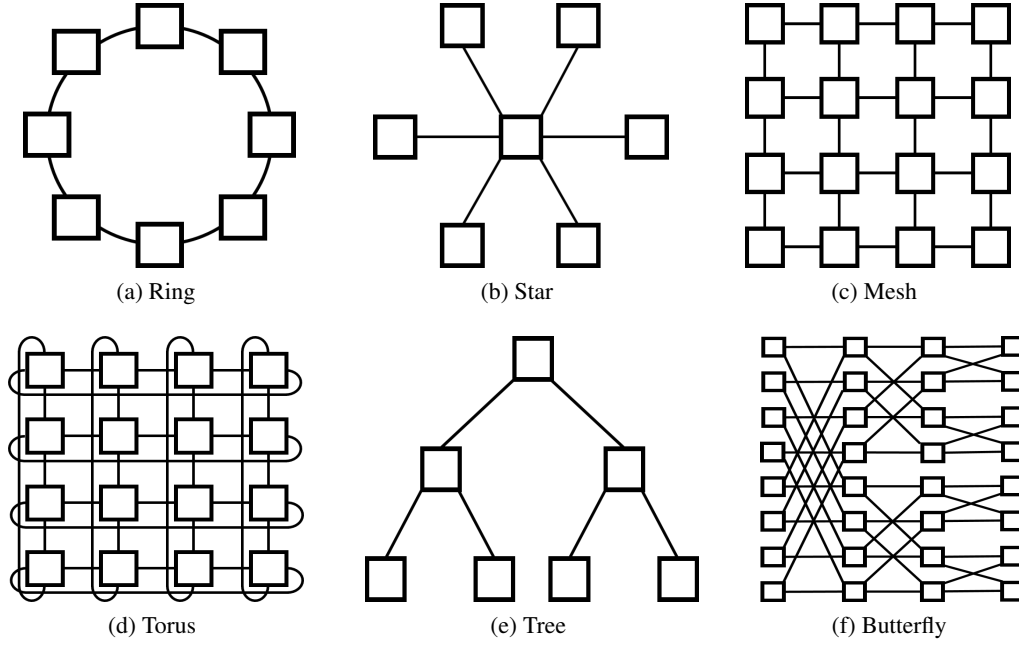


Figure 2.1: Network Topologies.

Each topology has its own characteristics and hence advantages and limitations, e.g., in terms of system performance, design complexity, design cost, fault tolerance, and bottleneck situations [14]. Let's consider two different topologies in NoCs as examples: ring and mesh.

The ring topology is among the simplest NoC topologies to design and implement. It consists of nodes connected in a circular manner, with each node directly linked to its two neighboring nodes. The regularity of the ring topology makes it easy to understand and analyze, simplifying the design process and reducing overall complexity. Consequently, the ring topology has lower design costs and requires fewer hardware resources compared to more complex topologies. However, the ring topology also has some limitations. Communication paths between non-adjacent nodes are longer, leading to potential congestion and resulting in higher latency and reduced throughput. Additionally, the ring topology suffers from limited fault tolerance. If a single node fails, it can disrupt the entire ring, affecting communication between certain nodes.

On the other hand, the mesh topology in NoCs offers significant advantages, mainly in terms of flexibility, efficiency in routing, and fault tolerance. With multiple direct paths between nodes, flits can take shorter and less congested routes, leading to reduced latency and improved communication performance. Additionally, the redundant connections in the mesh topology enhance fault tolerance, enabling the network to reroute communication through alternative paths in case of node failures, ensuring that data can still reach its destination. Moreover, the mesh topology is highly scalable, making it suitable for systems with a large number of processing elements. It can easily accommodate additional nodes without significantly affecting overall network performance.

However, these advantages come with certain limitations. The increased number of interconnections in the mesh topology results in higher design complexity, making the routing algorithms and network management more complex and challenging to implement. Furthermore, the mesh topology requires more hardware resources compared to simpler topologies like the ring, leading to higher costs and power consumption. Additionally, despite its flexible routing, the mesh topology may face congestion issues when multiple communication flows compete for resources.

The appropriate choice of topology depends on the specific requirements and constraints of the application to be executed in the network. Considering the previous comparison, the ring topology is an ideal choice for smaller systems with moderate communication demands, offering a simple and cost-effective solution. In contrast, the mesh topology is better suited for larger systems with higher communication demands and a requirement for fault tolerance. Therefore, network topology is a critical design aspect, and understanding the various options and their implications is essential for effective NoC design and implementation.

2.1.2 Routing Mechanism

In NoCs, routing refers to the process of transferring data packets from source to destination with a well-defined strategy. The routing mechanism then consists of a set of rules and algorithms implemented within the routers to determine the specific path and direction packets should follow as they traverse the network from the source to the destination [75]. To ensure the good behaviour of the network, routing mechanisms must provide certain guarantees, in particular deadlock-free and livelock-free transmissions [32]. Deadlock occurs when a packet does not progress in the network because it is permanently waiting for an event that cannot occur. Livelock, on the other hand, occurs when a packet travels indefinitely through the network and never reaches its destination. Clearly, those should never happen in a well-designed (real-time) NoC.

Several challenges must be considered when designing a routing mechanism for NoCs. These include the need to minimize communication latency and maximize throughput, parallelism, and communication channel utilization. The routing mechanism must be able to meet the high-bandwidth and low-latency communication requirements of modern systems and adapt to changes in communication patterns and workloads. To address these challenges, a wide range of routing mechanisms have been proposed for NoCs, including adaptive and deterministic approaches.

Adaptive routing mechanisms are designed to dynamically select the best path for packet transmission based on network conditions at run-time. These mechanisms continuously monitor factors such as network congestion and available resources to make routing decisions. Thus, they ensure that packets are directed along the least congested or fastest path. On the other hand, deterministic routing mechanisms rely on predetermined paths or routing tables to guide the forwarding of packets. These mechanisms follow a set of rules to determine the next hop for each packet based on factors such as destination address or network topology. When a deterministic routing mechanism guarantees that packets are routed over the minimum possible path from origin to destination, it is called minimal deterministic routing algorithm. One of the most popular minimal deterministic routing mechanisms is *Dimension Ordered Routing* (DOR) because it is simple, cheap (in terms

of implementation cost), and easy to implement. DOR is commonly used when routers in the network are connected together by a mesh or torus topology. There are several variants of DOR, *X-Y routing* is one of them [95]. In X-Y routing, a packet first travels horizontally on the X-axis of the mesh/torus, and then, once it reaches the X-coordinate of its destination router, it is routed vertically on the Y-axis of the network (see Figure 2.2).

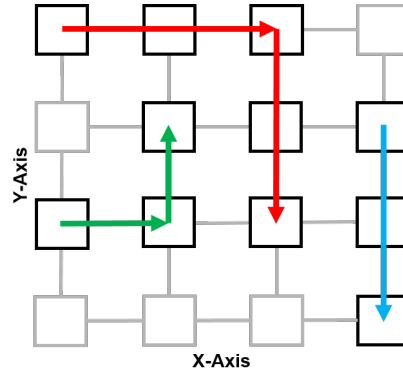


Figure 2.2: Examples of X-Y routing.

Each approach has its own characteristics and trade-offs. For example, by adapting to changing network conditions, adaptive routing mechanisms can achieve higher utilization, optimize packet delivery, and improve overall network efficiency. However, they suffer from low predictability and are susceptible to noise and interference. On the other hand, deterministic routing mechanisms provide predictable performance since the same path is always chosen for packets with the same characteristics (i.e., packets that have the same origin and destination nodes are always routed over the same path). However, they may not be as efficient in terms of communication resource utilization.

2.1.3 Switching Mechanism

To determine how data is transmitted from the origin node to the destination node through the network, NoCs rely on switching mechanisms. There are two basic approaches to data switching: circuit switching and packet switching. In circuit switching, the entire path that data must follow from its origin to its destination is determined and allocated before the data is sent. Packet switching, on the other hand, does not require the entire path from origin to destination to be reserved at once. Instead, the data is segmented, i.e., it is divided into multiple packets that can be transmitted independently through the network. Each individual packet contains information about the destination node and can therefore take a different route to reach it.

Packet switching techniques include but are not limited to *store-and-forward* and *wormhole switching techniques* [40]. In the store-and-forward technique, packets are received in their entirety by the router before being forwarded to the next router in their path (see Figure 2.3). However, this approach requires oversized buffers at each node, making it unattractive and impractical for embedded systems with severe area constraints [1].

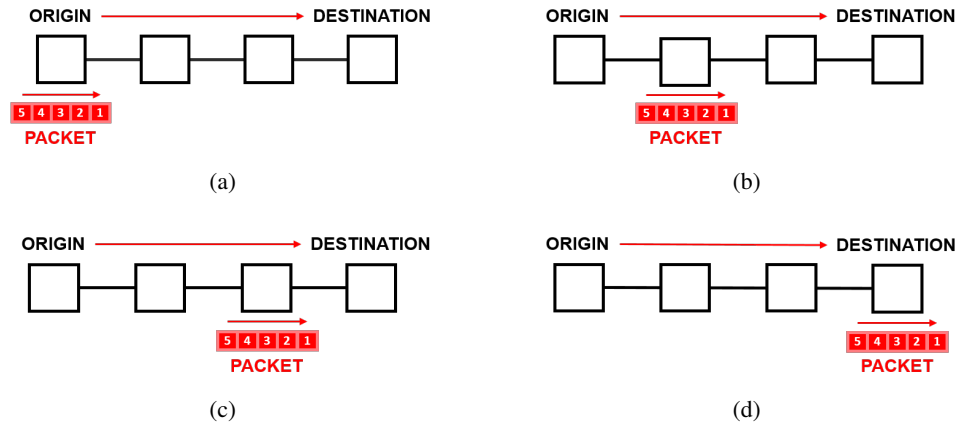


Figure 2.3: Example of Store-and-Forward technique.

In contrast, the wormhole switching technique requires smaller buffers and offers relatively low network latencies [61]. In wormhole switching, packets are split into multiple fixed-size elements, called flits, which are injected into the network sequentially. The first flit of the packet is called the header flit and the last is called the tail flit. The header and tail are the first and last flit of the packet injected into the network by the programming element, respectively. The header flit specifies the path to be followed, and the other flits follow one after another in a pipeline fashion (see Figure 2.4). While the use of extensive buffers on ports to store complete packets makes the store-and-forward technique unattractive to designers, wormhole switching is one of the most commonly used techniques in MPSoC because it reduces buffer requirements.

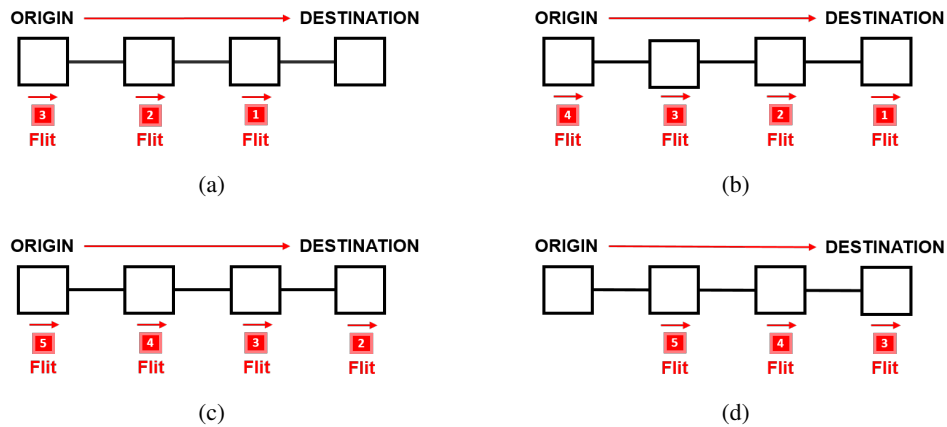


Figure 2.4: Example of Wormhole Switching technique.

2.1.4 Flow-Control Strategy

Flow-control strategies are an essential aspect of NoC designs, responsible for regulating the flow of flits within the network to prevent congestion and buffer overflow. The purpose of a flow-control strategy is to ensure that the transmitting device does not send more data to the receiving device

than it can handle. That is, flow control mechanisms are implemented to prevent new flits from entering routers whose buffers are already occupied by other flits [17].

In NoC designs, a widely adopted flow-control strategy is *credit-based flow control* (see Figure 2.5). This approach operates by assigning a certain number of credits to each port of a router, representing the number of slots available in the buffer associated with that port within the router. When a flit arrives at the router, it consumes one credit as it occupies a buffer slot. The router then sends an updated credit count to the upstream router, indicating the remaining available credits. To forward a flit to a downstream router, it must have sufficient credits available. If there are no credits left for a specific port, the flit must wait until a credit becomes available before it can be transmitted further.

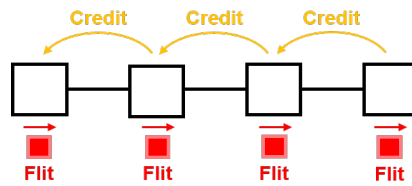
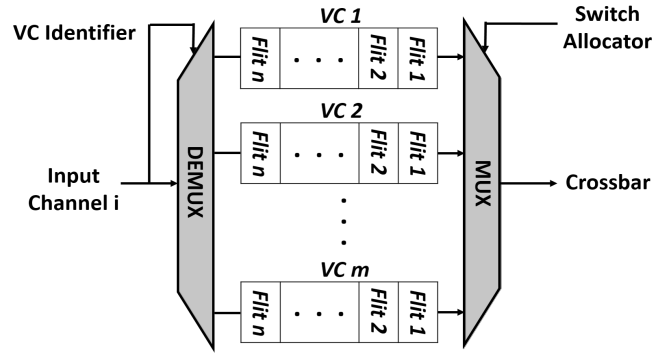


Figure 2.5: Conceptual view of credit-based flow control strategy.

Credit-based flow control is widely used in NoCs with wormhole switching mechanisms, but it is not the only option available. One alternative scheme is known as *back suction flow-control strategy* [22]. Unlike credit-based flow control, which relies on the use of credits to regulate the flow of flits, back suction adopts a different approach. In back suction flow control, when a router's output buffer becomes full, it sends a back suction signal to the upstream router. This signal requests the upstream router to temporarily stop sending additional flits to the congested output port. This action allows the congested router to alleviate buffer occupancy and create space for new flits to be injected into the network.

Credit-based flow control is well-suited for NoCs that require continuous communication flow with regulated buffer utilization. On the other hand, back suction flow control is suitable for scenarios where buffer size is limited, and preventing buffer overflow is critical, even if it results in temporary communication suspensions. The selection between these strategies depends on the specific requirements and constraints of the NoC application, as each strategy offers distinct advantages in managing flow and buffer resources.

To enhance the throughput of NoCs, additional hardware components like Virtual Channels (VCs) are embedded. With this approach, each physical channel in the network is partitioned into multiple VCs. VCs are basically buffers located in the input or output ports of each router, allowing flits entering from different ports to be stored in parallel (see Figure 2.6). VCs provide a way to multiplex multiple communication flows over the same physical link by allowing each flow to have its own dedicated VC in each router it reaches on its route toward its destination, leading to a more efficient use of the available bandwidth. Specifically, each VC can be exclusively used by a particular communication flow or shared by multiple communication flows.

Figure 2.6: Virtual Channels located in an input port i

While VCs are a popular solution to enhance the overall performance of NoCs, they come with some trade-offs. VCs significantly increase the silicon footprint and power consumption, making them more suitable for high-end applications. For systems with low power consumption and/or limited hardware resources, the use of VCs may not be practical or feasible. Therefore, the decision to use VCs in a NoC design depends on the specific application's requirements and the system's constraints.

2.1.5 Contention scenarios

In NoCs, multiple PEs may simultaneously attempt to transmit packets over the same communication channel. This situation is defined as a contention scenario and can lead to packet collisions, packet transmission delays, and buffer overflows if not properly managed. Specifically, two types of contentions can occur, namely direct and indirect contention. Direct contention occurs when two or more packets compete for the same resources, i.e., they simultaneously request a shared buffer or the same output port. As an example of direct contention, consider Figure 2.7a, in which three packets from three different communication flows compete for the same output port.

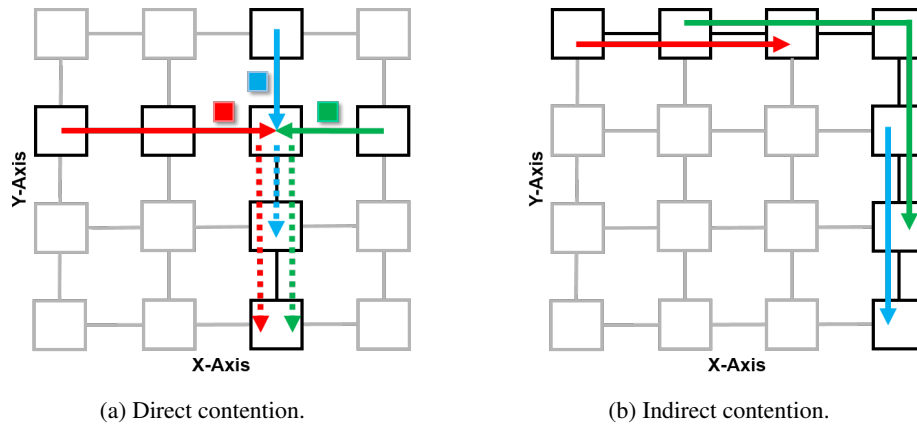


Figure 2.7: Example of contention scenarios.

On the other hand, indirect interference occurs when the transmission of one flow affects the progress of another flow with which it does not share some resources. See Figure 2.7b for an example. In this scenario, the red flow is blocked by the green flow, which is, in turn, blocked by the blue flow. The progress of the red flow is delayed by the transmission of the blue flow, despite the fact that they do not share any common resources. That is, the blue flow indirectly blocks the red flow's transmission.

Under contention scenarios, arbitration mechanisms decide which flit is sent first through each output port. In solutions relying on VCs, the flits winning the right to use the desired output ports are routed through them, while the remaining flits are stored in the router's VCs until their requests can be attended. Different VCs can be assigned various priorities, allowing flits buffered in higher-priority VCs to be transmitted before those in lower-priority ones.

Instead of relying on VCs or extensive buffers, alternative approaches utilize the concept of deflection to handle packets that do not win access to the requested output port in a contention scenario. These approaches are commonly known as deflection-based routing policies. In such, when multiple flits contend for the same output port, one flit is routed to the desired output port, while the remaining flits are deflected to alternative output ports. This enables efficient utilization of network resources and helps alleviate contention-related issues. Refer Figure 2.8 for an example.

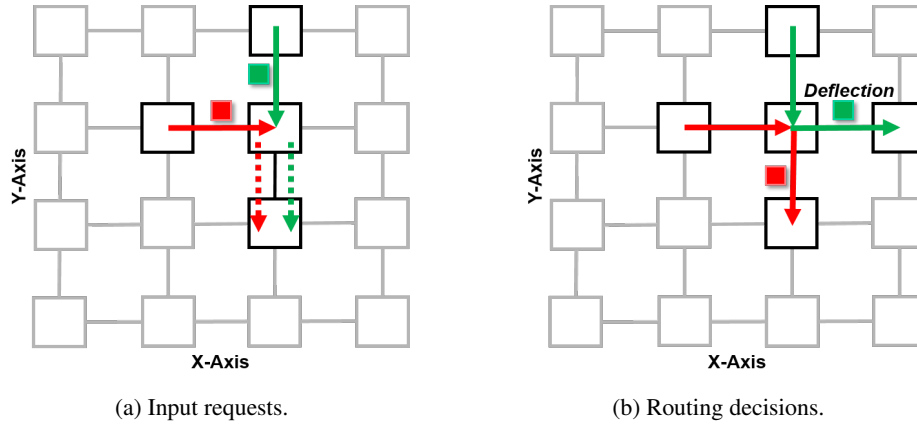


Figure 2.8: Example of deflection-based routing policy.

2.1.6 Arbitration Mechanisms

Several NoC designs proposed over the years avoid traffic contention by implementing pre-allocation strategies (e.g., by pre-allocating time slots or entire paths [34, 9, 69, 70, 84]). These solutions are called *contention-less approaches*. One popular strategy is *time-division multiplexing (TDM)*. TDM avoids conflicts between traffic flows by transmitting multiple packets over the same link, but assigning a fixed-length time slot to each of them.

Virtual circuit is another well-known method in NoCs. When using virtual circuits, the resources along the entire route of a packet are dynamically reserved before it is transmitted. In virtual circuit-based networks, a connection setup phase precedes the actual data transmission.

During this setup phase, the necessary resources (e.g., buffers, bandwidth, and routes) are reserved for the duration of the communication session or until the connection is explicitly terminated. The reservation of resources is based on the specific communication requirements and traffic conditions at the time of connection establishment. Once the virtual circuit is set up, the reserved resources are exclusively dedicated to the communication flow for the duration of the session. This helps ensure QoS guarantees and minimizes contention and congestion during data transmission. Upon completion of the communication session or when the connection is explicitly terminated, the previously reserved resources are released and become available for other communication flows.

In contrast, approaches that allow for contention scenarios are usually referred to as *contention-aware approaches* (e.g., [88, 20]). Contention-aware approaches use various arbitration mechanisms (e.g. *round-robin* or *priority-based arbitration policies*) to define precedence criteria when multiple packets request the same resource simultaneously. For example, in round-robin, access to the shared resource is granted to the requesting packets in a cyclic order. This ensures that each packet receives an equal share of the resource. In priority-based arbitration, on the other hand, access to the shared resource is granted based on the priority level of the communication flows to which the requesting packets belong. Requests from high-priority packets are processed before those from lower-priority packets. This can ensure that important tasks are completed quickly, but it can also result in low-priority requests being blocked for a long time.

2.2 NoCs for Real-Time Systems

Over the years, several NoCs integrating various design aspects have been proposed for real-time systems. The first group [34, 9, 84, 3] relies on time-triggered routing and arbitration protocols. These NoCs are referred to as contention-less solutions. In such systems, the timing properties of communication flows are isolated by assigning pre-computed transmission slots to each flow. However, this approach suffers from some limitations. Specifically, it requires complete knowledge of system specifications at configuration time and does not support changes in system workload during run-time. Although these solutions provide robust guarantees for real-time systems, they do not support dynamic systems. Most modern applications, however, are inherently dynamic. Therefore, in this work, we focus on contention-aware NoCs, which are designed to support dynamic systems. This section provides a brief overview of selected real-time and contention-aware NoC architectures based on different approaches.

2.2.1 NoCs Based on Wormhole Switching with VCs

One widely recognized family of real-time NoCs relies on wormhole switching with VCs. Figure 2.9a illustrates an example of a generic VCs-based router. Embedded commercial off-the-shelf (COTS) multi/many-core chips such as Kalray MPPA [20] and Tilera Tile CPUs [92] rely on VCs-based NoC architectures. VCs form the backbone of the most common real-time NoCs arbitration policies that have been studied in the real-time systems literature, e.g., IDAMC [88], *fixed priority*

preemption based wormhole switching [79] and *EDF based wormhole switching* [62]. Moreover, the most popular real-time NoC architectures [67, 88, 59] rely on these concepts to achieve bounded worst-case traversal time. Below are some examples of NoCs in this group.

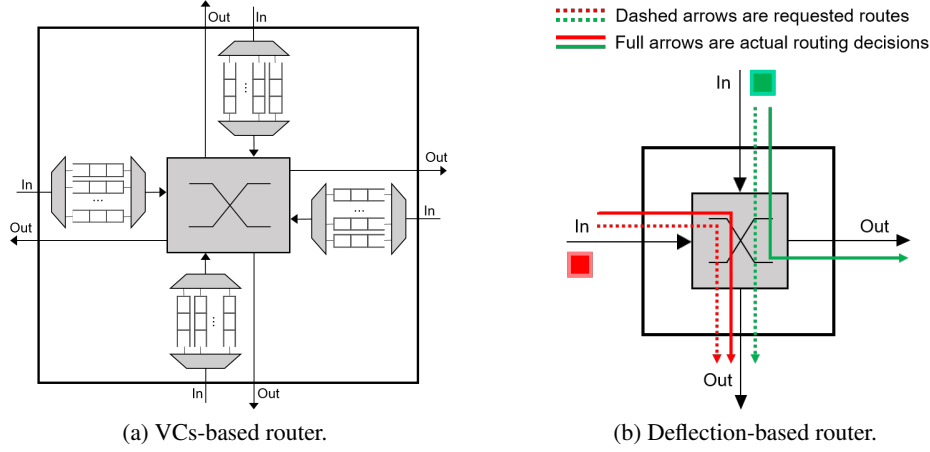


Figure 2.9: Example of routers based on different strategies.

ProNoC: ProNoC [59] is a NoC architecture that provides flexible and configurable connectivity, allowing developers to dynamically configure and reconfigure the network topology through programmable routers. It adopts the most advanced NoC features such as VCs. It also implements low-latency routing techniques and offers a range of routing algorithms. ProNoC can be easily extended or modified to accommodate new components or subsystems without requiring significant changes to the overall system architecture. By reconfiguring the network on-the-fly, developers can allocate more bandwidth to critical components, balance communication load, or adjust network parameters to optimize performance, power consumption, and reliability based on the specific requirements of the system or application being executed.

IDAMC: IDAMC [88] is a NoC architecture designed to provide a communication infrastructure that efficiently supports mixed-criticality systems, i.e., systems in which multiple components or tasks with different levels of criticality coexist and interact [12]. IDAMC supports routers with up to eight ports, enabling the creation of diverse network topologies such as mesh, ring, and star configurations. It supports wormhole switching and multiple virtual channels. IDAMC uses traffic segregation mechanisms to separate critical and non-critical communication flows, e.g., independent virtual channels are used for different criticality levels. IDAMC also supports adaptability to handle dynamic changes in critical system requirements. It allows reconfiguration of network resources at runtime, ensuring that critical tasks remain unaffected by changes and non-critical tasks can effectively use available resources without compromising safety-critical operations. To ensure service guarantees, IDAMC implements the Back Suction flow-control strategy [22]. That strategy ensures that critical traffic is prioritized to receive higher bandwidth, lower latency, and minimal interference from non-critical traffic.

CONNECT: CONNECT [67] is a versatile and adaptable on-chip communication solution, designed to provide flexible interconnectivity. By offering customizable features such as variable input and output ports, flit sizes, and multiple virtual channels with adjustable flit buffer depths, CONNECT enables tailored configurations to suit specific needs. This solution allows for customization of the network topology, enabling integration into various system architectures, accommodating various communication patterns, and meeting performance requirements. In addition, CONNECT supports two flow control mechanisms: credit-based flow control and peek flow control. To optimize performance, CONNECT employs a single-stage router pipeline, reducing network latency, minimizing hardware resource consumption, and simplifying flow control processes.

Wormhole switching with VCs leads to powerful NoC infrastructures that provide different levels of QoS and ensure efficient packet traversal times and network bandwidth utilization while guaranteeing bounded worst-case traversal times. The fact that the approach relies heavily on VCs to provide timing guarantees, makes these solutions more expensive in terms of hardware resource utilization and power consumption than alternative designs like contention-less or deflection-based networks which will be discussed later in this chapter.

2.2.1.1 Worst-Case Network Latency Analysis for Wormhole-Based NoCs

Various timing analyses for wormhole-based NoCs can be found in the literature (e.g., [4, 73, 72]), each with unique strengths and limitations, designed to address diverse aspects of network performance and behavior.

In [27], a recursive calculus method for worst-case analysis was proposed, originally designed for SpaceWire network (which is high-speed on-board satellite network characterized by small-scale interconnects with limited links and data flow). The method is scalable and can potentially be adapted for NoCs.

Some works, including [51, 54, 80, 50], provide timing analysis techniques for priority-based wormhole switching NoCs. In [80], Shi et al. introduce an analysis of worst-case network latency for a real-time fixed-priority preemptive wormhole-based NoC with dedicated VCs for each priority level, considering both direct and indirect interference. Several variations of that approach were proposed over the years [82, 23, 88, 18, 74], for instance, handling the case where several flows share the same priority [53], changing the routing policy to Earliest Deadline First (EDF) [62] or supporting communication flows with different criticality levels [13, 42]. However, the complexity of the NoCs design and their routing policies makes their analysis complex, pessimistic, and "hard-to-get-right", as evidenced by issues that were recently discovered and exposed in [94, 93, 41, 63]. [30, 31] proposed a worst-case timing analysis of wormhole-based NoCs using Network Calculus.

2.2.2 NoCs with a Deflection-Based Routing Policy

For systems with critical constraints in terms of area and/or power consumption, deflection-based NoCs [58, 49, 90, 77, 78, 25] have shown promise as a solution for the future generation of MP-SoCs. Typically, deflection-based NoCs rely on simpler routing policies and have lower design complexity than NoCs solutions based on VCs or extensive buffers (see Figure 2.9b). As a result, their implementation cost and power consumption are lower, and deriving temporal and throughput analysis is easier compared to the previously discussed designs. However, the approach is more susceptible to livelocks situations and there is very limited effort for real-time systems. Indeed, until very recently, and as far as we know, the only real-time NoC based on the concept of deflection that exhibits predictable timing behavior, and comes up with a complete analysis of the worst-case communication time of packets is HopliteRT NoC. Introduced by Wasly et al. in [90], HopliteRT is a slightly modified version of Hoplite¹ that makes it compatible with real-time systems requirements. It ensures an upper bound on the packets WCTT thanks to modifying the routing policy of Hoplite and introducing a traffic injection regulation protocol.

HopliteRT is a buffer-less design that ensures predictable and easily analyzable timing behaviors. Furthermore, it is well-suited for dynamic workloads and can be effectively used in most systems, including those with low power consumption and limited resource availability. However, HopliteRT does not provide different levels of QoS to communication flows. Additionally, flits from the same packet may arrive at their destination in a different order than they were injected at the source, resulting in out-of-order delivery. Finally, due to the network topology and routing policy supported in HopliteRT, flits may traverse multiple routers twice in the worst-case scenario.

2.2.3 Other NoCs architectures

MANGO: MANGO [10] is a clockless NoC architecture built using asynchronous circuits for both its links and routers. This design approach helps reduce power consumption and mitigate clock-related issues such as clock skew and synchronization problems, which are common in synchronous systems. MANGO provides robust QoS by supporting two types of communication services: Guaranteed Services (GS) ensure that specific communication channels have dedicated bandwidth and bounded latency, making them ideal for time-sensitive applications, while Best-Effort (BE) Services handle traffic without strict guarantees on bandwidth or latency, suitable for less critical data transfers. The asynchronous design, while advantageous in many ways, can lead to complications in packet reordering, as packets may arrive out of sequence. To maintain the sequence integrity of data packets, MANGO includes mechanisms for reordering packets at their destination. However, implementing packet reordering requires sophisticated logic, which increases the complexity of the router design and adds latency. Moreover, while MANGO offers QoS features that can support time-sensitive applications, it is not explicitly designed as a real-time NoC, that is, its primary design goal is not to meet real-time constraints. Therefore, applications

¹Hoplite is a buffer-less deflection-based NoC infrastructure proposed in [49] that reduces the NoC features to their bare minimum and hence decreases considerably the network analysis complexity and implementation cost (in terms of hardware resources utilization).

with hard real-time requirements might need additional guarantees that are not inherently provided by MANGO.

On-chip ring designs: Panić et al. propose in [66] several time-predictable single and multi-ring designs with low hardware complexity for shared-memory multicore processors. These solutions support priority arbitration policies and include a mechanism to regulate the rate at which packets are injected into the ring. Additionally, the designs employ Time Division Multiple Access (TDMA) to manage link access. The TDMA scheme divides the link access into slots assigned to each node, ensuring that the slots receiving a packet from a source to a destination are synchronized. This synchronization ensures predictability and helps derive an upper bound on the worst-case traversal time of packets. However, ring topologies typically face scalability issues. Communication latency increases linearly with the number of nodes because, as more nodes are added, the average distance a packet must travel grows. Additionally, the mechanism used to regulate the rate at which packets are injected into the ring can lead to under-utilization of network bandwidth. That is, it may result in portions of the available bandwidth being unused, thus not leveraging the network's full capacity. On the other hand, in [21], the authors propose a low-cost dual-ring communication infrastructure designed for a real-time heterogeneous multicore stream processing architecture, specifically targeting Software Defined Radio (SDR) applications. This infrastructure provides both bandwidth and latency guarantees. One key feature of this design is its ability to allow tasks to utilize the reserved but unused bandwidth allocated to other tasks, thereby enhancing the average throughput at the application level. The infrastructure utilizes two separate, oppositely directed rings: one dedicated to data transmission and the other to credit transfers between tiles. Similar to the ring-based designs from [66], this solution suffers from the limitations of the ring topology.

HopliteBuf: HopliteBuf [28] is a recent solution for in-order packet delivery at destination inspired by a deflection based routing policy. HopliteBuf uses large buffers within routers to ensure the in-order packet delivery property and eliminate the need for deflection. This decision allows PEs to inject more flits to free output ports, which has a positive impact on flits's injection times. The buffers implemented in HopliteBuf are stall-free, avoiding interruptions or delays that could change the order of packets during transmission, thus ensuring that packets arrive at their destinations in exactly the same order in which they were injected at their sources. The buffer size in HopliteBuf depends on the specific applications running on the network. Thus, if more communication flows must be transmitted through the network, or even if their sources and destinations change, larger buffers are likely to be needed to avoid buffer overflows and hence loss of flits during their transmissions. It was shown in [33] that to support realistic communication loads, buffers in HopliteBuf must be oversized. Even with considerably large buffers, that is, 128-deep buffers in each router, buffer overflowed, resulting in 26% of the flits never reaching their destination. This necessity for large buffers in each router leads to increased resource and power requirements for the NoC.

2.3 HopliteRT NoC: Overview

Among the NoC architectures available in the literature, we found HopliteRT particularly attractive because its features closely align with the objectives of this research. HopliteRT served as the starting point for our work, and in this section, we delve into its properties and behavior in more detail.

2.3.0.1 System Model

HopliteRT assumed a system of N programming elements (PEs) labeled $\{\pi_1, \dots, \pi_N\}$ connected to a NoC of N routers labeled $\{R_1, \dots, R_N\}$, where each PE π_q is connected to router R_q with coordinates (x_q, y_d) in a 2D-grid representation of the NoC. Each PE π_q injects a set of n_q communication flows defined by $F_q = \{f_{q,1}, f_{q,2}, \dots, f_{q,n_q}\}$ into the network. Each communication flow $f_{q,i}$ is defined by a 3-tuple $\langle T_{q,i}, R_{q,i}^o, R_d^d \rangle$ as follows. Each flow $f_{q,i}$ generates a potentially infinite number of packets with a minimum inter-arrival time $T_{q,i}$ clock cycles. That is, $T_{q,i}$ is the minimum duration between the generation of two packets by $f_{q,i}$. Each packet has a single flit, which is injected at the origin router $R_{q,i}^o$ at coordinates $(x_{q,i}^o, y_{q,i}^o)$ and traverses the network to reach its destination router $R_{q,i}^d$ at coordinates $(x_{q,i}^d, y_{q,i}^d)$.

2.3.0.2 Network Topology and Router Architecture

In HopliteRT, the routers are connected according to a torus topology of dimension $S_x \times S_y$, where S_x and S_y represent the number of routers on the horizontal (i.e., X-axis) and vertical dimensions (i.e., y-axis), respectively (see Figure 2.10a). Each router of HopliteRT has three input ports labeled N , W , and PEi , and three output ports labeled S , E , PEo (see Figure 2.10b). The input ports N and W and the output ports S and E are used for communication between the routers. Ports S and E of each router are connected to ports N and W of the routers located directly south and east of them in the network, respectively (see Fig. 2.10a). The remaining ports, i.e., PEi and PEo , are connected to the programming element that communicates with the router. The port PEi is used by PE to inject flits into the router, while the port PEo is used by flits to exit the network.

2.3.0.3 Routing Policy

HopliteRT implements a modified version of X-Y routing. That is, a flit that has been injected into the network first travels east along the X-axis until it reaches a router with the same X coordinate as its destination (see Figure 2.11a, where R_o and R_d represent the origin and destination routers, respectively, and R_m is the router with the same X coordinate as its destination). Once in that router, the flit turns south and travels along the Y-axis until it reaches its destination (see Figure 2.11b). HopliteRT's routing policy differs from X-Y routing in that it allows flits to be deflected to the east while traveling south (see Figure 2.12). A deflected flit must then travel along the X-axis until it reaches the same router where it was deflected and resume its journey south.

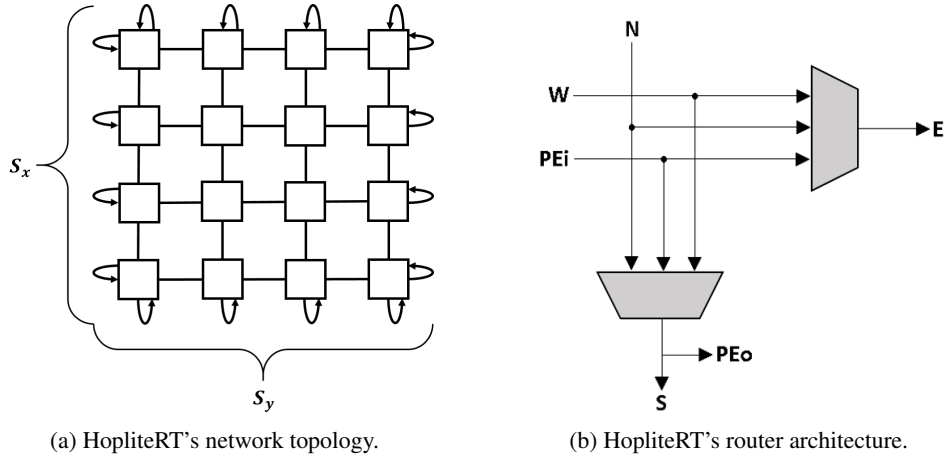


Figure 2.10: HopliteRT design.

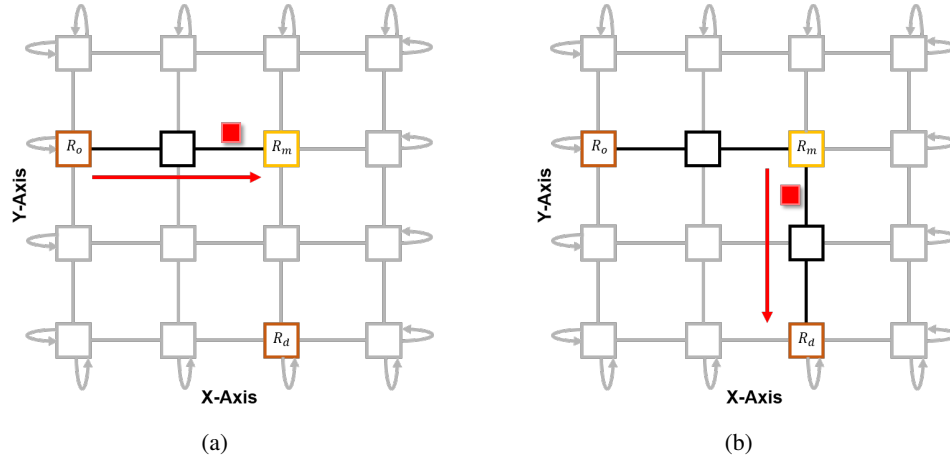


Figure 2.11: Example of HopliteRT's routing policy.

Specifically, flits can enter a router via ports N , W , and PEi (see Figure 2.10b). Flits entering a router by the ports W and PEi can request output ports E or S , while flits coming from port N can only request port S . Then, there is no conflict scenario over port E . That is, flits coming from port W and requesting port E travel freely along the X-axis. However, if a flit enters a router via port W , it may conflict with another flit coming from port N if both request port S at the same time. In this conflict scenario, HopliteRT always gives the highest priority to use port S to the flit coming from port W and the flit entering from port N is deflected east regardless of its final destination. After a contention scenario, a deflected flit must travel along the X-axis until reaching the same router where it was deflected. Consistently, it enters that router via port W and, therefore, is assigned the highest priority the next time it requests port S . Finally, flits coming from port PEi always have the lowest priority to use port S , and must wait inside PEs until there is a clock cycle in which no other flit requests those ports. HopliteRT's routing policy is summarized in Table 2.1.

Table 2.1: Routing Table of HopliteRT

Case	Input requests	Routing decisions	Explanation
1	$W \rightarrow E + N \rightarrow S$	$W \rightarrow E + N \rightarrow S$	No contention.
2	$W \rightarrow S + N \rightarrow S$	$W \rightarrow S + N \rightarrow E$	Conflict over port S. $W \rightarrow S$ request wins. N flit is deflected.
3	$N \rightarrow S + PE \rightarrow S$	$N \rightarrow S$	PE cannot inject its flit.
4	$N \rightarrow S + PE \rightarrow E$	$N \rightarrow S + PE \rightarrow E$	No contention.
5	$W \rightarrow E + PE \rightarrow S$	$W \rightarrow E + PE \rightarrow S$	No contention.
6	$W \rightarrow E + PE \rightarrow E$	$W \rightarrow E$	PE cannot inject its flit.
7	$W \rightarrow S + PE \rightarrow S$	$W \rightarrow S$	PE cannot inject its flit.
8	$W \rightarrow S + PE \rightarrow E$	$W \rightarrow S$	Not allowed. PE cannot inject its flit.

Example. A red flit entering a router R_m via port W requests port S at the same time as a green flit entering via port N (see Figure 2.12a and case 2 in Table 2.1). Since port W has the highest priority for using port S , the red flit continues its route to port S , while the green flit is deflected to port E (see Figure 2.12b).

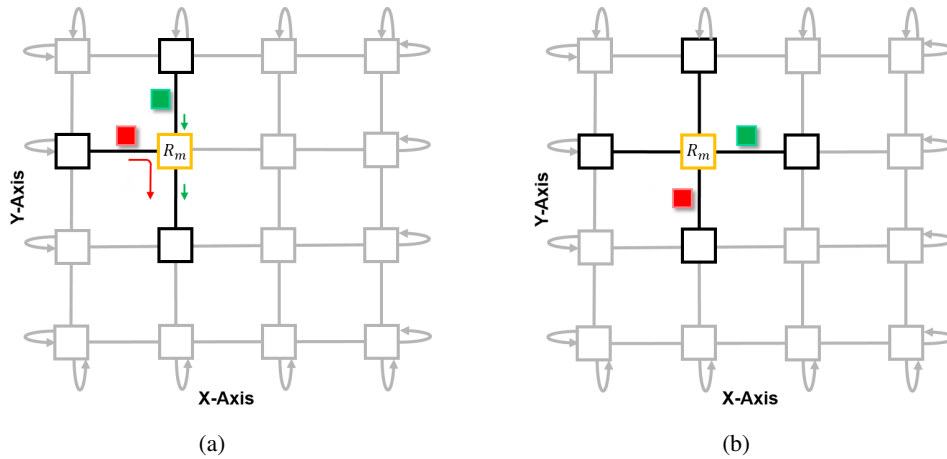


Figure 2.12: Example of a contention scenario in HopliteRT.

2.3.0.4 Traffic Injection Regulation Protocol

HopliteRT implements a traffic injection regulation protocol on port PEi of each router to limit the number of flits the PE can send in bursts on the network. This prevents some PEs from being blocked indefinitely because other PEs are using the bandwidth unfairly. HopliteRT's traffic injection regulation protocol is implemented with a leaky bucket that contains two cascaded counters on each PE π_q : (i) a rate counter and (ii) a token counter. The rate counter overflows every θ_q cycles, where θ_q is the flit injection period of the PE π_q . Then, the token counter is incremented every overflow until it reaches a maximum value σ_q . The value of the token counter determines the maximum number of flits that PE can inject consecutively. The value of the token counter is decremented each time a flit is injected into the network by π_q .

2.3.0.5 Worst-case Communication Time Analysis

The WCCT $wcct_i$ of a flit of a packet of flow $f_{q,i}$ is composed of the WCIT $wcit_i$ and WCTT $wctt_i$. The WCIT is the maximum number of clock cycles between the generation of a packet by a PE and the injection of its last flit into the network. The WCTT is then the maximum number of clock cycles required for a flit to traverse the network from origin to destination. The WCCT of a flit of a packet of flow $f_{q,i}$ is then given by

$$wcct_i = wcit_i + wctt_i \quad (2.1)$$

To derive a bound on the WCIT, $\Gamma_{q,i}^C$ is defined as the set of flows that conflict with the injection of a flit of flow $f_{q,i}$ by π_q . Then, $\sigma(\Gamma_{q,i}^C)$ and $\rho(\Gamma_{q,i}^C)$ define the cumulative burst length and injection rate of the conflicting flows, respectively. Then, [90, 91, 28] prove that if $\rho(\Gamma_{q,i}^C) < 1$ (i.e., the cumulative injection rate of the conflicting flows is less than 1 flit/cycle), eventually there will be available clock cycles, such that the injection of a packet of flow $f_{q,i}$ will not be blocked indefinitely. Thus, the WCIT $wcit_i$ of a flit of flow $f_{q,i}$ injected by π_q , assuming $\rho(\Gamma_{q,i}^C) < 1$ is given by

$$wcit_i = (\theta_{q,i} - 1) + B_i, \quad (2.2)$$

with

$$B_i = \left\lceil \frac{\sigma(\Gamma_{q,i}^C)}{1 - \rho(\Gamma_{q,i}^C)} \right\rceil \quad (2.3)$$

where

$$\sigma(\Gamma_{q,i}^C) = \sum_{f_{m,j} \in \Gamma_{q,i}^C} \sigma_{m,j} \quad (2.4)$$

and

$$\rho(\Gamma_{q,i}^C) = \sum_{f_{m,j} \in \Gamma_{q,i}^C} \frac{1}{\theta_{m,j}} \quad (2.5)$$

On the other hand, the WCTT $wctt_i$ of a flit transmitted over a $S_x \times S_y$ HopliteRT NoC from origin router $R_{q,i}^o$ at coordinates $(x_{q,i}^o, y_{q,i}^o)$ to destination router $R_{q,i}^d$ at coordinates $(x_{q,i}^d, y_{q,i}^d)$ is given by (in clock cycles)

$$wctt_i = h_x^i(x_{q,i}^d, y_{q,i}^d) + h_y^i(x_{q,i}^d, y_{q,i}^d) + (h_y^i(x_{q,i}^d, y_{q,i}^d) \times S_x) + 2 \quad (2.6)$$

where $h_x^i(x_{q,i}^d, y_{q,i}^d)$ and $h_y^i(x_{q,i}^d, y_{q,i}^d)$ are the number of hops the flit does on the X- and Y-axis from its origin PE to its destination PE, assuming zero contention, i.e., when the flit suffers no deflection. Then, $h_x^i(x_{q,i}^d, y_{q,i}^d) = (x_{q,i}^d - x_{q,i}^o + S_x) \bmod S_x$ and $h_y^i(x_{q,i}^d, y_{q,i}^d) = (y_{q,i}^d - y_{q,i}^o + S_y) \bmod S_y$. Note that, mod is the modulo operator. The term $h_y^i(x_{q,i}^d, y_{q,i}^d) \times S_x$ represents the total cost of

potential deflections. According to HopliteRT's routing policy, a flit can be deflected at most $h_y^i(x_{q,i}^d, y_{q,i}^d)$ times, and each such deflection increases the packet's traversal time by S_x hops. The two additional cycles counted in the above equation correspond to the one clock cycle needed to inject the flit into the network at its origin PE and the one clock cycle needed to hop between the router and the PE at its destination ².

2.4 FPGA Platform in the Scope of NoCs

In parallel with the increasing complexity of SoCs, the enhanced capabilities of FPGA platforms and their flexibility to implement any digital functionality by programming reconfigurable elements have made FPGAs a viable alternative to ASICs for custom SoC development. FPGAs enable the development of systems with a high degree of parallelism at a relatively low cost. Indeed, FPGAs can be reprogrammed and reconfigured as needed, allowing changes to be made to the system without requiring extensive redesign or reassembly. Therefore, FPGAs facilitate rapid prototyping, shorten development time and reduce process costs compared to ASICs. This flexibility can be especially useful for systems that need to be upgraded or modified to support new features or technologies.

A complete MPSoC can be implemented on an advanced FPGA (e.g., [60]). However, the number of processors that can be embedded is limited by the capacity of such platforms, i.e., by the number of reconfigurable elements of FPGAs. As a result, most FPGAs do not provide enough resources to embed complex NoC designs with a large number of processing elements. Therefore, any new NoC design targeting FPGA-based MPSoCs must be simple, compact, and cost-effective (in terms of hardware requirements) while enforcing strong real-time guarantees and ensuring low power consumption and high system performance.

2.5 Chapter Summary

The chapter offered a comprehensive exploration of different approaches to designing NoCs. It began by delving into the design features of NoCs and provided an overview of existing literature on NoCs for real-time systems. The chapter then specifically focused on two popular families of NoC solutions: wormhole switching with virtual channels and deflection-based NoCs. Through a thorough analysis of the strengths and weaknesses of each approach, the chapter provided a deeper understanding of their suitability for real-time applications. Additionally, it discussed the role of FPGA platforms in the context of NoCs, highlighting their significance in implementing and evaluating NoC solutions. With all the relevant concepts and approaches introduced, the upcoming chapters will present the contributions of this dissertation. Each chapter will delve into specific solutions that contribute to the overall findings and advancements of this work.

²Note that, in Appendix A, we demonstrate that the analysis presented in [90, 91] for bounding packets' WCIT is not safe. A corrected version of the analysis is introduced in Section 3.4.3.

Chapter 3

Revisiting HopliteRT for Improved Real-Time Performances

As previously mentioned, two commonly employed strategies for developing NoC solutions in real-time systems are wormhole switching with VCs and deflection-based routing policy. Among these approaches, deflection-based solutions have proven to be more suitable for systems with power and/or area constraints.

In the preceding section, we analyzed HopliteRT, highlighting its advantages and limitations. As previously discussed, HopliteRT treats all packets identically, i.e., it does not allow to associate different priorities, and consequently, different quality of service, to distinct communication flows. Additionally, HopliteRT suffers from long penalties for deflecting flits, which directly affects packets' worst-case traversal time. To illustrate this behavior, let's consider the following example.

Example. Consider a 4x4 HopliteRT NoC (see Figure 3.1). Consider also two flits (red and green flits) entering router R_m via ports W and N , respectively, and requesting port S at the same time (see Figure 3.1a). According to HopliteRT's routing policy, in this conflict scenario, the red flit wins the right to use port S , while the green flit is deflected to port E (see Figure 3.1b and case 2 in Table 2.1). After being deflected, the green flit will traverse through S_x routers to return to the same router in which it was initially deflected, making no progress towards its destination during these S_x additional hops (see from Figure 3.1c to Figure 3.1f).

Motivated by these considerations, in this chapter, we propose a novel real-time buffer-less NoC architecture called HopliteRT* that keeps the advantages of HopliteRT while improving its real-time capabilities. HopliteRT* supports a new priority-based routing policy built upon the concept of deflection that uses the properties of a fully scalable topology. In particular, it targets systems with power and/or area constraints and limited resource availability, e.g., mid-range FPGA platforms. HopliteRT* is the first step towards achieving the ultimate goal of this research, that is to design cost-efficient NoC architectures for real-time embedded systems that offer both deterministic timing behavior and flexibility to accommodate a wide range of platforms and application requirements.

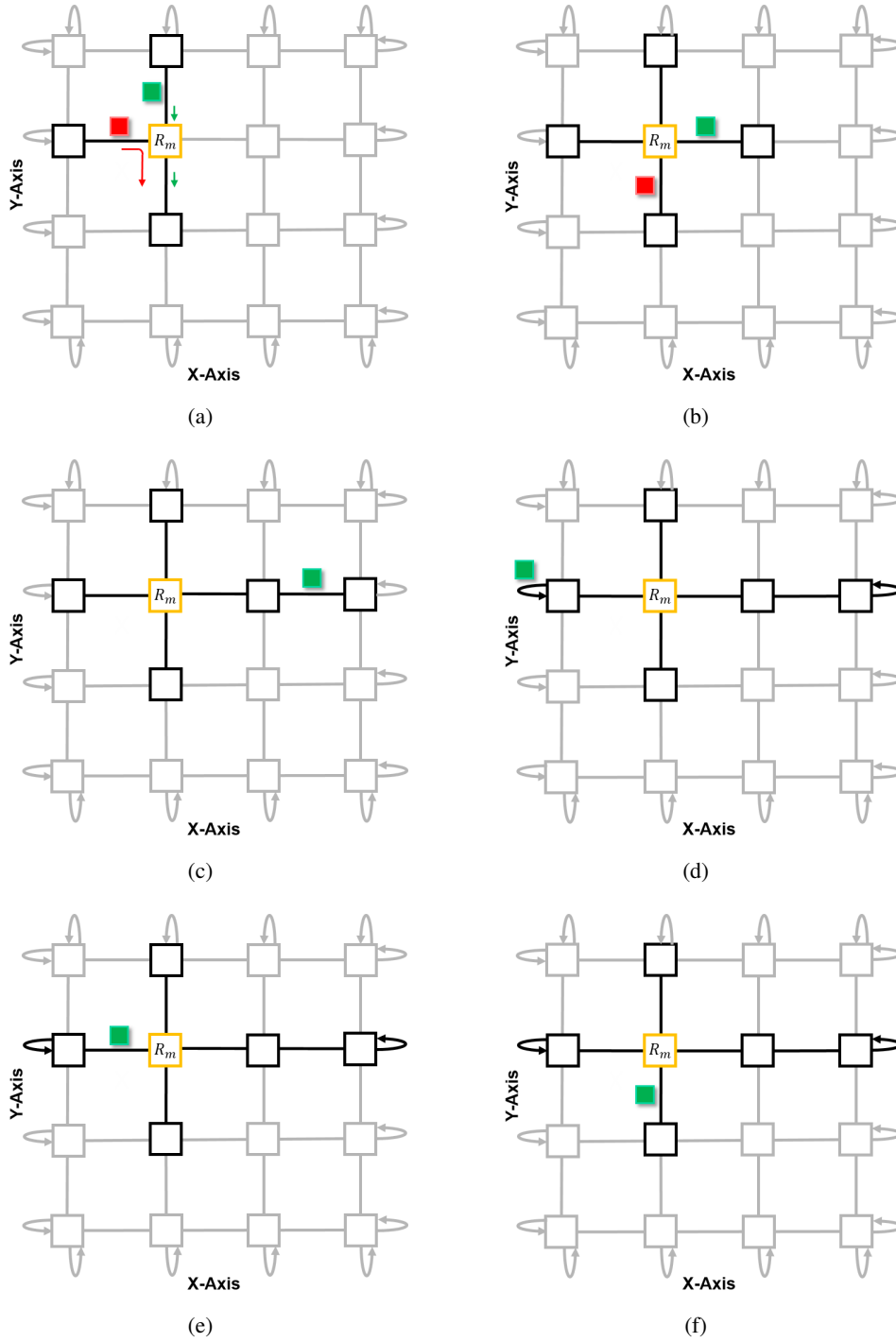


Figure 3.1: Example of the penalties for deflecting flits in HopliteRT.

The chapter follows the following structure. In Section 3.1, we present the system model. HopliteRT* is introduced in Section 3.2, where we provide details on the routing policy, network topology, and router architecture. Section 3.3 discusses various properties and behaviors of the solution, while Section 3.4 derives an upper-bound on the WCCT of packets traversing HopliteRT* NoC. Finally, in Section 3.5, we present and discuss various experimental results.

The content of this chapter was previously published in the following papers:

- **Y. Ribot González** and G. Nelissen. HopliteRT*: Real-Time NoC for FPGA. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3650-3661, 2020, doi:10.1109/TCAD.2020.3012748.
- **Y. Ribot González**, and G. Nelissen. Design and implementation of an FPGA-based NoC for Real Time Systems. In Proceedings of the Work-in-Progress Session (ECRTS'19): 1-3, 2019.

3.1 System Model

We assume a system composed of N programming elements (PEs) labeled $\{\pi_1, \dots, \pi_N\}$ connected to a NoC of N routers labeled $\{R_1, \dots, R_N\}$, where each PE π_q is connected to router R_q with coordinates (x_q, y_d) in a 2D-grid representation of the NoC. Each PE π_q injects a set of n_q communication flows defined by $F_q = \{f_{q,1}, f_{q,2}, \dots, f_{q,n_q}\}$ into the network. Each communication flow $f_{q,i}$ is defined by a 5-tuple $\langle C_{q,i}, T_{q,i}, P_{q,i}, R_{q,i}^o, R_{q,i}^d \rangle$ as follows. Each flow $f_{q,i}$ generates a potentially infinite number of packets with a minimum inter-arrival time $T_{q,i}$ clock cycles. That is, $T_{q,i}$ is the minimum duration between the generation of two packets by $f_{q,i}$. Each packet is then split into $C_{q,i}$ flits, which are injected sequentially at the origin router $R_{q,i}^o$ at coordinates $(x_{q,i}^o, y_{q,i}^o)$ and traverse the network to reach their destination router $R_{q,i}^d$ at coordinates $(x_{q,i}^d, y_{q,i}^d)$. Each flow is assigned a priority $P_{q,i}$, which can be 0 or 1, corresponding to low and high priority, respectively. We denote the sets of high and low priority flows released in π_q as $hp_q = \{f_{q,i} \mid P_{q,i} = 1\}$ and $lp_q = \{f_{q,i} \mid P_{q,i} = 0\}$. We assume that all the routing information is encoded in each flit of the packet, i.e., there is no distinction between header, body, or tail flits. The routing information is composed of the coordinates of the destination and one bit encoding the priority. The number of bits used for the destination coordinate depends only on the size of the network, e.g., 64-bits wide flits traversing a 4x4 NoC use only 4 bits to encode their destination coordinates, in which case 59 bits are left for the payload. The utilization of a flow $f_{q,i}$ is defined as $U_{q,i} = \frac{C_{q,i}}{T_{q,i}}$. Then, the total utilization of a PE π_q is found by summing the utilization of all flows in F_q , i.e., $U_q = \sum_{f_{q,i} \in F_q} U_{q,i}$.

3.2 HopliteRT* Design

3.2.1 Priority-Based Routing Policy

The basics of the routing policy of HopliteRT* is the same as that of HopliteRT, which we recall is a modified version of X-Y routing. As in HopliteRT, a flit that is injected in HopliteRT* first travels east along the X-axis until it reaches a router with the same X coordinate as its destination. Once in that router, the flit turns south and travels along the Y-axis until it reaches its destination. HopliteRT*'s routing policy differs from X-Y routing in that if two flits request the same port S at the same time, it "deflects" one of the flits to the port E . A deflected flit must then travel

along the X-axis until reaching the same router as it would have if it had not been deflected. It will then request the port S again and continue traveling along the Y-axis. The main difference between HopliteRT and HopliteRT* is that HopliteRT* bases its decision of which flit to deflect on priorities that are associated to those flits. A low priority flit will be the one that is deflected if it contend with a higher priority flit. If contending flits have the same priority then the same arbitration policy than HopliteRT is applied.

Specifically, flits can enter a router via ports N , W , and PEi (see Figure 3.4). Flits entering a router by the ports W and PEi can request output ports E or S , while flits coming from port N can only request port S . Then, there is no conflict scenario over port E . That is, flits coming from port W and requesting port E travel freely along the X-axis. However, if a flit enters a router via port W , it may conflict with another flit coming from port N if both request port S at the same time. In this conflict scenario, in HopliteRT, the flits coming from port W always have the highest priority to use port S . Instead, HopliteRT* applies a routing policy based on a scheme with two priority levels: Low and High., where low priority flits will never be permitted to deflect high priority flits. That is, if in the above conflict scenario, the flit coming from port N has a high priority and the flit entering by port W has a low priority, then the high priority flit wins the right to use port S , and the flit coming from port W is deflected east without regard to its final destination. In this way, HopliteRT* ensures that low priority flits can never interfere with the communication time of high priority flits. In any other case, the routing policy is the same as in HopliteRT (refer to Section 2.3.0.3 and Table 2.1). That is, HopliteRT* always gives any flit coming from port W the highest priority to use port S . To implement this new routing policy, the priority of each flit is encoded in its most significant bit.

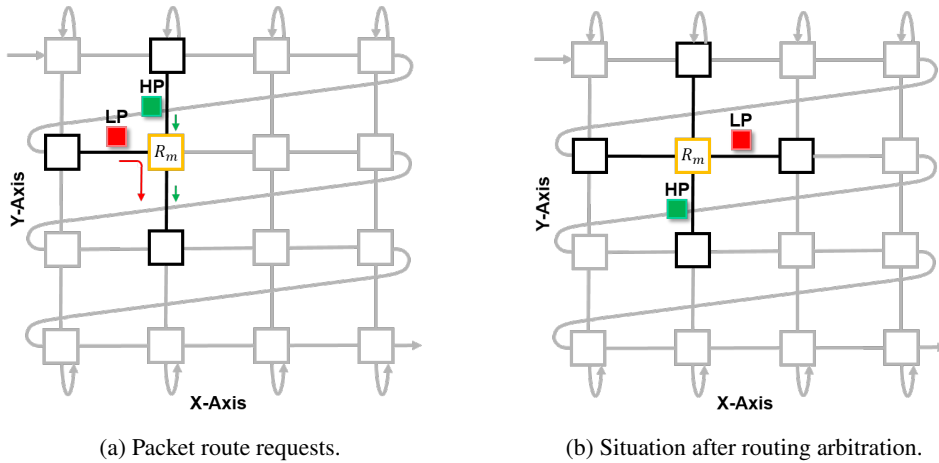


Figure 3.2: Example of HopliteRT*'s priority-based routing policy.

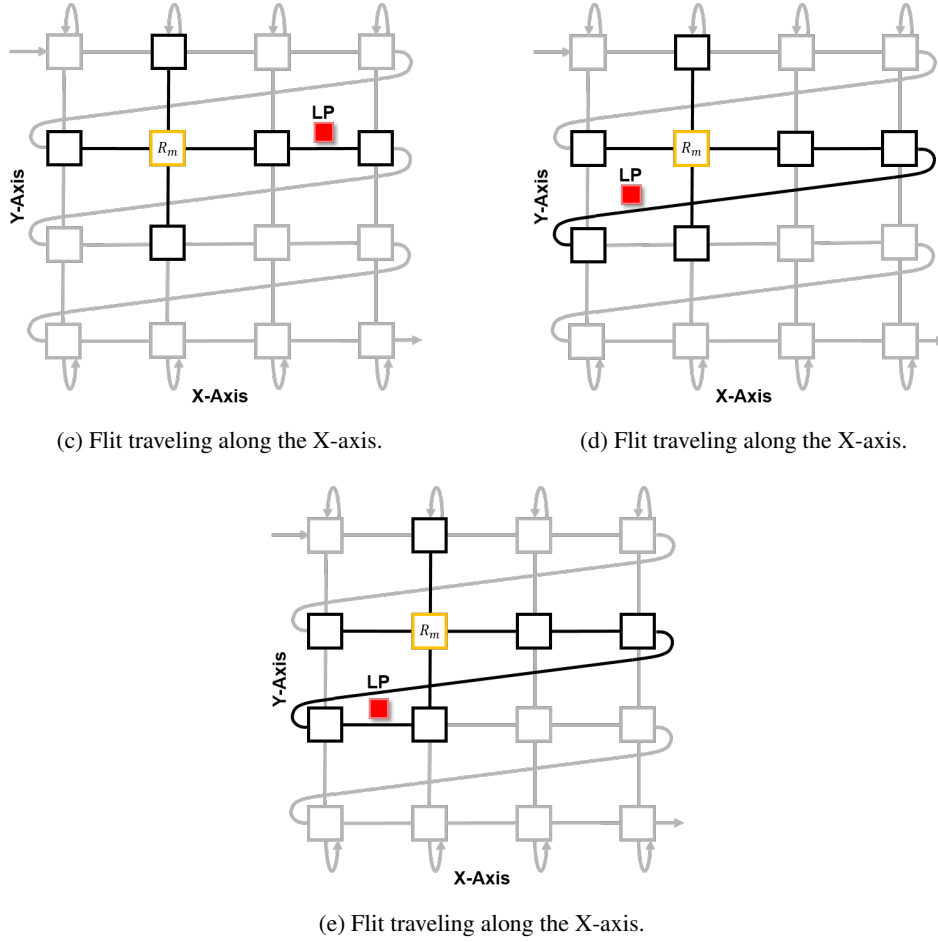


Figure 3.2: Example of HopliteRT*'s priority-based routing policy (cont.).

Example. Consider a 4x4 HopliteRT* NoC (see Figure 3.2) and a flit of a low priority packet (red packet) entering router R_m via port W. In R_m , the red flit requests port S at the same time as a flit of a high-priority packet (green packet) entering R_m via port N (see Figure 3.2a). In this conflict scenario, the high priority flit (i.e., the green flit) wins the right to use port S, while the low-priority flit (i.e., the red flit) is deflected to port E (see Figure 3.2b and case 2 in Table 3.1). The deflected flit then travels along the X-axis until it arrives at the router it originally requested before the deflection (see from Figure 3.2c to Figure 3.2e).

Finally, flits coming from port PE_i always have the lowest priority to use port S, and must wait inside the PEs until there is a clock cycle in which no other flit requests the desired port. HopliteRT*'s routing policy is summarized in Table 3.1. The text highlighted in red indicates the difference between HopliteRT's and HopliteRT*'s routing policies.

3.2.2 Ensuring Progress: Circulant Topology on Time Determinism

Even though it looks beneficial, the priority-based routing policy we just proposed is in fact extremely inefficient, and in fact even detrimental to timing predictability, if we keep the same net-

work topology as HopliteRT (i.e., a torus topology). Consider the red flit in Figure 3.2. Since that flit was deflected, it will hop through S_x routers before entering again in the same router in which it was initially deflected. That is, the flit did not progress at all toward its destination after those S_x additional hops. If the red packet has a low priority, it may be deflected again and again in the same router, thus, never experiencing any progress. Therefore, the WCTT of high priority flits remains unchanged (only their average-case traversal time is potentially reduced), but more importantly, the WCTT of low priority flits is not bounded anymore.

Table 3.1: Routing Table of HopliteRT*

Case	Input requests	Routing decisions	Explanation
1	$W \rightarrow E + N \rightarrow S$	$W \rightarrow E + N \rightarrow S$	No contention.
2	$W^{HP} \rightarrow S + N^{LP} \rightarrow S$ $W^{HP} \rightarrow S + N^{HP} \rightarrow S$ $W^{LP} \rightarrow S + N^{LP} \rightarrow S$ $W^{LP} \rightarrow S + N^{HP} \rightarrow S$	$W \rightarrow S + N \rightarrow E$ $W \rightarrow S + N \rightarrow E$ $W \rightarrow S + N \rightarrow E$ $W \rightarrow E + N \rightarrow S$	High priority $W \rightarrow S$ always wins; High priority $W \rightarrow S$ always wins; $W \rightarrow S$ wins over $N \rightarrow S$ of same priority; A low priority request never wins over a high priority one.
3	$N \rightarrow S + PE \rightarrow S$	$N \rightarrow S$	PE cannot inject its flit.
4	$N \rightarrow S + PE \rightarrow E$	$N \rightarrow S + PE \rightarrow E$	No contention.
5	$W \rightarrow E + PE \rightarrow S$	$W \rightarrow E + PE \rightarrow S$	No contention.
6	$W \rightarrow E + PE \rightarrow E$	$W \rightarrow E$	PE cannot inject its flit.
7	$W \rightarrow S + PE \rightarrow S$	$W \rightarrow S$	PE cannot inject its flit.
8	$W \rightarrow S + PE \rightarrow E$	$W \rightarrow S$	Not allowed. PE cannot inject its flit.

We prevent this lack of progress by changing the network topology. We connect routers according to a ring circulant topology defined as $(N; g_1, g_2)$ (see Figure 3.3a), where N is the total number of routers, $g_1 = 1$, g_2 is a divisor of N , and g_1, g_2 are the *generatrices* of the network. The total number of generatrices determines the number of neighbouring routers to which each router can send and receive flits to/from. The value of a generatrix represents the distance between two connected routers. For example, if $g_x = 1$ each router is connected to the next router on the ring, and if $g_x = 4$ each router is connected to the router four hops away on the ring. Thus, since $g_1 = 1$, in HopliteRT* all routers are connected by a single unidirectional ring through their ports W and E (as shown by red links in Figure 3.3a). Each router is also connected to routers g_2 hops away on the ring via ports N and S (see green and black links in Figure 3.3a).

The circulant network $(N; 1, g_2)$ may be represented as a $S_x \times S_y$ grid-based network (see Figure 3.3b), where S_x and S_y are the number of routers on the horizontal and vertical dimensions, respectively. Then, the values of S_x and S_y are given by $S_x = g_2$ and $S_y = \frac{N}{g_2}$. The red, green, and black links in Figure 3.3a correspond to the red, green, and black links in Figure 3.3b, respectively. The grid-based network shown in Figure 3.3b is merely an abstract representation of the circulant topology to facilitate later discussion of PEs and router coordinates, and not an actual physical mapping of the routers in HopliteRT*. In the actual implementation, there is no long wire and thus no transmission latency issues that affect the maximum clock frequency of the network. The long wire connecting port E of the router (3,3) to port W of the router (0,0) in Figure 3.3b is just the result of representing a 3D structure in a 2D drawing. All wires connecting successive routers on the ring have the same length (see Figure 3.3a). The length of the wires can be reduced by map-

ping the routers appropriately during the placement and routing step on the final chip. Similarly to what is done with a torus topology.

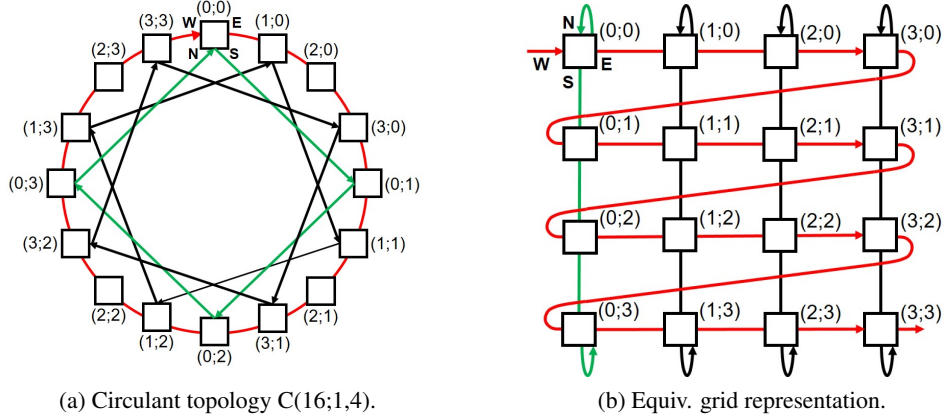


Figure 3.3: HopliteRT* network topology.

Example. Figure 3.3a shows the circulant network with parameters $(16; 1, 4)$. In Figure 3.3b, we provide its equivalent representation as a 4x4 grid-based network. The main unidirectional ring (red link in Figure 3.3a) corresponds to the rows of the grid (see Figure 3.3b), where the E port of the last router in case y is connected to the W port of the first router in case $(y + 1) \bmod S_y$. Similarly, the bypasses (green and black in Figure 3.3a) correspond to the links on the columns of the grid where the last router in a column is connected to the first router in that same column.

Note that the circulant topology does not suffer from the same limitations as the traditional ring topology as it is fully scalable. Indeed, the generatrices allow the NoC to have as much bandwidth as with a torus topology. Thanks to this new topology, after a contention scenario, a deflected flit travels along the X-axis until it reaches the same router it would have reached if it had not been deflected. That is, the packet always progresses toward its destination, even if it was deflected (see from Figure 3.2c to Figure 3.2e).

3.2.3 Router Architecture

The new topology of HopliteRT* requires slightly modify the router design. Indeed, originally, in the particular case where two flits arrive at the same instant in the same router (via ports W and N) and that router is their destination, HopliteRT would send one flit to the PE and would deflect the other to the port E . The deflected flit would then travel around the whole row in order to reach its destination router once more and finally exit the network. If the same approach was adopted in HopliteRT*, because of the change of topology, the deflected flit may have had to travel around the entire network (instead of just the current row) before reaching its destination router for the second time. This situation causes a remarkable and unacceptable increase in the WCTT of the deflected flit. We aim to solve this issue by allowing the PE connected to the router to read both packets simultaneously. For this reason, each router of HopliteRT* has three input ports labeled N , W , and PEi , and four output ports labeled S , E , PEo_1 , and PEo_2 (see Figure 3.4). The input

ports N and W and the output ports S and E are used for communication between the routers. Ports S and E of each router are connected to ports N and W of the routers located directly south and east of them in the network, respectively (see Figure 3.3b). The remaining ports, i.e., PEi , PEo_1 , and PEo_2 , are connected to the programming element that communicates with the router. The port PEi is used by PE to inject flits into the router, while the ports PEo_1 and PEo_2 are used by flits to exit the network (instead of just port PEo as in HopliteRT). Note that, PEo_1 shares its wires with port S and PEo_2 shares its wires with port E . Sharing output ports with the inputs of the PE slightly increases the complexity of the logic in the router, but it avoids the implementation of expensive multiplexers.

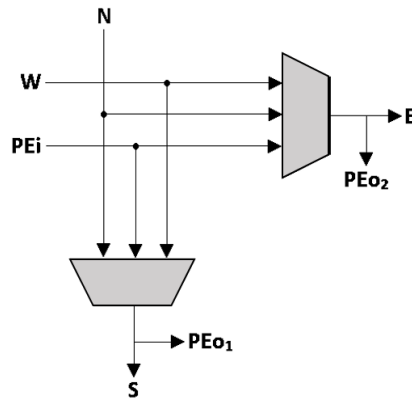


Figure 3.4: HopliteRT* router architecture.

In HopliteRT*, the number of flits entering a router always matches the number of flits leaving that router. In other words, there is always an available output port for incoming flits to use, either the desired output port or an alternative one. However, once flits reach their destination, to prevent the rate of outgoing flits from exceeding the capacity of the PEs to receive flits, we consider that each PE has two FIFO queues, one for each output port, which are sufficiently large to prevent data loss or overwriting of information. We connect ports PEo_1 and PEo_2 to these buffers, ensuring that each PE is able to serve two requests per cycle (i.e., one from PEo_1 and one from PEo_2). As a result, PEs can accept flits as quickly as they desire to leave the network, effectively preventing the occurrence of back-pressure in the network.

We also consider that each PE has two FIFO queues (which may be implemented in software or hardware), one per priority level, for flits that are pending to be injected into the network. High-priority flits are injected first in the network. Flits of low-priority flows are injected only when the high-priority FIFO queue is empty. That is, high-priority flits do not suffer delay due to low-priority flits sent by the same PE.

However, we assume that each flow may have at most one packet in the FIFO queue pending to be injected into the network at any time instant. After injecting a packet, a new packet from the same flow may be stored in the FIFO to be injected. In other words, we assume that $\forall f_{q,i}, T_{q,i} \geq wcit_i$.

3.3 Discussion

3.3.1 Traffic Injection Discussion

HopliteRT* assumes that there is no regulation of traffic injection on PEs, i.e., PEs can inject flits as fast as possible. According to HopliteRT*'s routing policy, a PE π_q can inject a flit of flow $f_{q,i}$ on port S or E of its associated router R_q only if there is no flit coming from a neighboring router requesting the same output port as PE at the same time (see Table 3.1).

Example. Consider three routers R_p , R_m , and R_q (red, yellow, and green routers, respectively) of a 4×4 HopliteRT* NoC (see Figure 3.5). Suppose that a packet of flow $f_{p,i}$ with $C_{p,i} = 26$ flits and $x_{p,i}^o \neq x_{p,i}^d$ (red packet) releases within PE π_p (see Figure 3.5a). Assume that no flit enters router R_p via port W and requests port E . π_p then starts injecting flits at port E of R_p as fast as possible (see Figure 3.5a). Now suppose that while π_p is injecting flits, a yellow packet of flow $f_{m,i}$ with $C_{m,i} = 30$ flits and $x_{m,i}^o \neq x_{m,i}^d$ releases within π_m and a green packet of flow $f_{q,i}$ with $C_{q,i} = 20$ flits and $x_{q,i}^o \neq x_{q,i}^d$ releases within π_q (see Figure 3.5b). Then, π_m and π_q are prevented from injecting flits into the network because ports E of R_m and R_q , respectively, remain occupied by the red flits coming from R_p . π_m can inject its first flits only when port E of R_m becomes free, i.e., when π_p has no more flit to inject (see Figure 3.5c). Note that, π_q is also blocked by flits coming from π_m . Therefore, π_q has no chance to inject flits until π_p and π_m have injected all their flits (see Figures 3.5d and 3.5e).

This approach maximizes network throughput but can lead to unfair network bandwidth usage. PEs located in regions of low congestion may allocate network bandwidth indiscriminately, preventing other PEs from injecting flits into the network. Furthermore, it makes the communication system vulnerable to Denial-of-Service attacks, which can disrupt network operations and compromise its reliability. In Chapter 6, we introduced a new strategy for regulating traffic injection based on network congestion to address these issues.

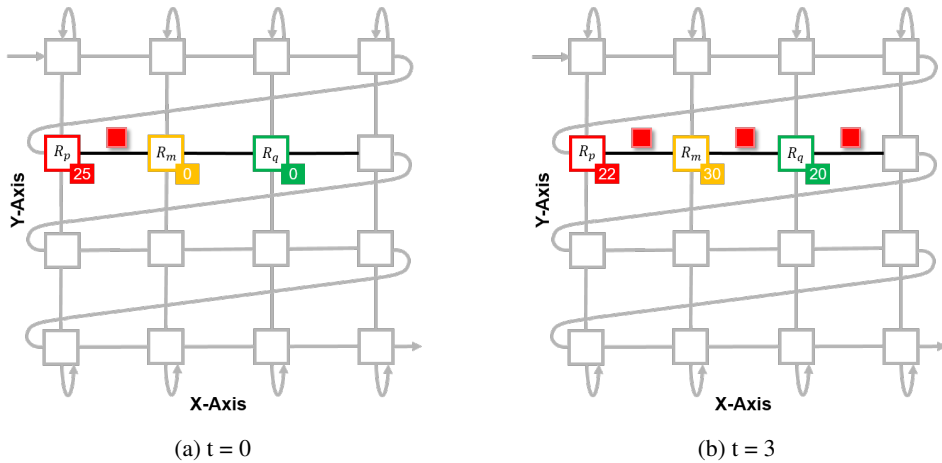


Figure 3.5: Example of traffic injection in HopliteRT* NoC.

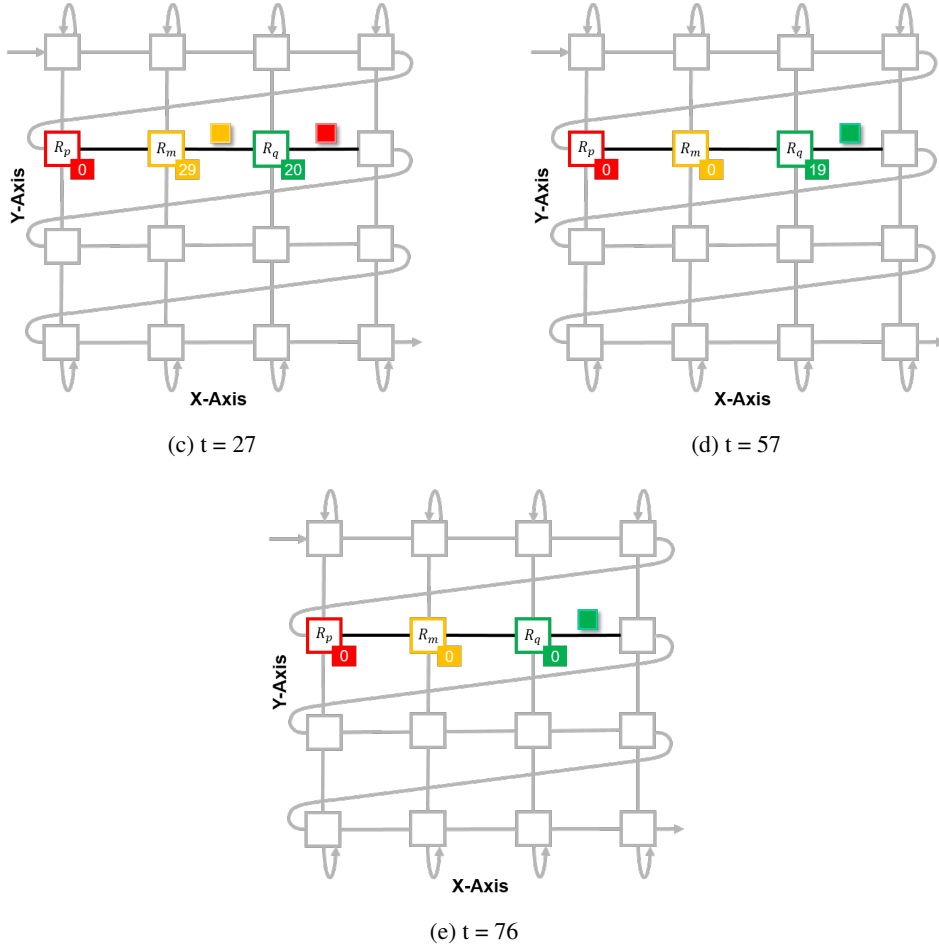


Figure 3.5: Example of traffic injection in HopliteRT* NoC (cont.).

3.3.2 Livelock and Deadlock Situations

HopliteRT* guarantees freedom from deadlock, as any flit entering a router and requesting an output port may be routed to the desired port or deflected to an alternative port. In either scenario, the flit progresses toward its destination without interruption.

On the other hand, HopliteRT*'s routing policy also guarantees livelock-free transmissions. The circulant topology ensures that flits always progress toward their destination routers even when they are deflected. Once at the destination routers, flits always exit the network by using the output ports PEo_1 and PEo_2 .

3.4 Worst-case Communication Time Analysis

In this section, we propose a WCCT analysis for HopliteRT*. We assume that $T_{q,i} \geq C_{q,i}$. The WCCT of a flit of flow $f_{q,i}$ is then defined as:

$$wcct_i = wcit_i + wctt_i, \quad (3.1)$$

where $wcit_i$ is the worst-case injection time and $wctt_i$ is the worst-case traversal time of a flit of flow $f_{q,i}$.

3.4.1 Bound on the Worst-Case Traversal Time

We start by deriving an upper-bound on the WCTT of a flit of flow $f_{q,i}$ (see Theorem 1).

Theorem 1. The WCTT of a flit of flow $f_{q,i}$ is given by the equation

$$wctt_i = n_{hops}^i(x_{q,i}^d, y_{q,i}^d) + n_{def}^i(x_{q,i}^d, y_{q,i}^d) \times c_{def}, \quad (3.2)$$

where $n_{hops}^i(x_{q,i}^d, y_{q,i}^d)$ is the number of hops made by the flit from its origin PE until its destination PE, assuming zero contention (i.e., that no flit is ever deflected in the network); $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ is the maximum number of deflections suffered by the flit on its route until its destination; and c_{def} is the maximum number of cycles by which the journey of a flit may be increased due to a single deflection in one of the routers.

Proof. According to HopliteRT*'s routing policy, when a flit is injected into the origin router, it initially travels along the X-axis of the network until it reaches a router with the same X-coordinate as its destination router. Upon reaching this router, it requests port S and starts its journey along the Y-axis. During this journey, it always covers the same distance of $n_{hops}^i(x_{q,i}^d, y_{q,i}^d)$ hops from its origin router to its destination router, assuming that it does not contend with other flits over the output ports. However, during the vertical traversal, the flit may conflict for port S with other entering flits that also request the same output port at the same time. In such cases, the flit under analysis can be deflected to port E . The potential for deflection exists whenever the flit requests port S , resulting in a total of $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ deflections while traveling south. After each deflection, the flit must travel along the X-axis until it reaches the same router it would have arrived at if it had not been deflected. This extra number of hops is always the same for each deflection as, according to the routing policy, the flit will not contend with any other flit while traveling along the X-axis. This additional travel along the X-axis corresponds to c_{def} extra hops, with each deflection increasing the traversal time by that amount. Therefore, the WCTT of a flit of flow $f_{q,i}$ is calculated by summing the number of hops $n_{hops}^i(x_{q,i}^d, y_{q,i}^d)$ the flit does from origin to destination router assuming zero contention, and the total additional cost introduced by $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ potential deflections, which is $n_{def}^i(x_{q,i}^d, y_{q,i}^d) \times c_{def}$ hops. Thus, we have $wctt_i = n_{hops}^i(x_{q,i}^d, y_{q,i}^d) + n_{def}^i(x_{q,i}^d, y_{q,i}^d) \times c_{def}$, concluding the proof of the lemma. $\square$

The term $n_{hops}^i(x_{q,i}^d, y_{q,i}^d)$ is defined as

$$n_{hops}^i(x_{q,i}^d, y_{q,i}^d) = h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) + 2, \quad (3.3)$$

where $h_r^i(x_{q,i}^d, y_{q,i}^d)$ is the number of hops made by the flit on the ring (i.e., the number of hops along the X-axis) until reaching its destination, assuming a zero contention. Similarly, $h_b^i(x_{q,i}^d, y_{q,i}^d)$ is the number of hops on the bypasses (i.e., number of hops along the Y-axis) until its destination,

assuming no contention. Simply speaking, $h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d)$ is the Manhattan distance between the origin and destination routers, where $h_r^i(x_{q,i}^d, y_{q,i}^d)$ and $h_b^i(x_{q,i}^d, y_{q,i}^d)$ are the X- and Y-distance, respectively. The two hops added at the end of Equation (3.3) account for the one cycle needed to inject the flit into the network and the one cycle needed by the destination PE to read that flit.

In the following, we prove upper-bounds for $h_r^i(x_{q,i}^d, y_{q,i}^d)$ (Lemma 1), $h_b^i(x_{q,i}^d, y_{q,i}^d)$ (Lemma 2), $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ (Lemmas 3 and 5) and $c_{def}(x_d, y_d)$ (Lemma 6).

Lemma 1. The number of hops on the ring by a flit of flow $f_{q,i}$ to reach a destination $(x_{q,i}^d, y_{q,i}^d)$ in a zero contention network is given by $h_r^i(x_{q,i}^d, y_{q,i}^d) = (x_{q,i}^d - x_{q,i}^o + S_x) \bmod S_x$.

Proof. According to HopliteRT*'s routing policy, each flit travels first through the ring from the origin router at coordinate $(x_{q,i}^o, y_{q,i}^o)$ until it reaches a router with the same X coordinate as the destination. According to the topology presented in Figure 3.3b, the number of hops $h_r^i(x_{q,i}^d, y_{q,i}^d)$ on the ring is thus, $(x_{q,i}^d - x_{q,i}^o)$ when $x_{q,i}^d \geq x_{q,i}^o$ and $(x_{q,i}^d - x_{q,i}^o + S_x)$ when $x_{q,i}^d < x_{q,i}^o$. That is, $h_r^i(x_{q,i}^d, y_{q,i}^d) = (x_{q,i}^d - x_{q,i}^o + S_x) \bmod S_x$. $\square$

Lemma 2. The number of hops on bypasses by a flit of flow $f_{q,i}$ to reach a destination $(x_{q,i}^d, y_{q,i}^d)$ in a zero contention network is given by $h_b^i(x_{q,i}^d, y_{q,i}^d) = (y_{q,i}^d - y_{q,i}^{o'} + S_y) \bmod S_y$ where

$$y_{q,i}^{o'} = \begin{cases} y_{q,i}^o & \text{when } x_{q,i}^d \geq x_{q,i}^o \\ y_{q,i}^o + 1 & \text{when } x_{q,i}^d < x_{q,i}^o \end{cases} \quad (3.4)$$

Proof. Remember that a bypass in Figure 3.3a corresponds to a link of a column of grid representation of the network in Figure 3.3b. Let $y_{q,i}^{o'}$ be the Y coordinate of the router at which the flit stops traveling on the ring and starts using bypasses (i.e., the first router with the same X coordinate as the destination). Then, according to the router numbering shown in Figure 3.3b, $y_{q,i}^{o'} = y_{q,i}^o$ when $x_{q,i}^d \geq x_{q,i}^o$ and $y_{q,i}^{o'} = y_{q,i}^o + 1$ when $x_{q,i}^d < x_{q,i}^o$, and the number of hops $h_b^i(x_{q,i}^d, y_{q,i}^d)$ on the Y-axis is $y_{q,i}^d - y_{q,i}^{o'}$ when $y_{q,i}^d \geq y_{q,i}^{o'}$ and $y_{q,i}^d - y_{q,i}^{o'} + S_y$ otherwise. That is, $h_b^i(x_{q,i}^d, y_{q,i}^d) = (y_{q,i}^d - y_{q,i}^{o'} + S_y) \bmod S_y$. $\square$

The maximum number of deflections $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ that a flit may suffer until its destination $(x_{q,i}^d, y_{q,i}^d)$ differs for low and high priority flits. We analyze both cases in Lemmas 3 and 5.

Lemma 3. The maximum number of deflections $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ suffered by a flit of a low priority flow with destination $(x_{q,i}^d, y_{q,i}^d)$ is upper-bounded by $h_b^i(x_{q,i}^d, y_{q,i}^d)$.

Proof. According to HopliteRT*'s routing policy. A low-priority flit entering from port W or N may always be deflected. Therefore, a low-priority packet may be deflected as many times as it may try to use a bypass, i.e., $h_b^i(x_{q,i}^d, y_{q,i}^d)$ times. $\square$

Lemma 4. A flit of a high-priority flow cannot be deflected in two successive routers on the same column (i.e., two routers directly connected to one another by their ports S and N).

Proof. We prove this lemma by contradiction. Consider that the high-priority flit is deflected in two successive routers R_m and R_p on the same column (i.e., port S of R_m is connected to port N of R_p). Because the flit is deflected in R_p , then, according to the routing policy of HopliteRT*, it must have entered by port N of R_p . That is, it must have exited R_m by port S . Since the flit left R_m by port S , it means that it was not deflected in R_m , thereby leading to a contradiction. $\square$

Lemma 5. The maximum number of deflections $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ suffered by a flit of a high priority flow with destination $(x_{q,i}^d, y_{q,i}^d)$ is upper-bounded by $\left\lfloor \frac{h_b^i(x_{q,i}^d, y_{q,i}^d)}{2} \right\rfloor$.

Proof. According to HopliteRT*'s routing policy, a high-priority flit can only be deflected when it enters a router by port N . That is, it can only be deflected when it travels along bypasses. Note that, a flit can only use bypasses that belong to the same column (i.e., through routers with the same X coordinate as its destination).

By definition of $h_b^i(x_{q,i}^d, y_{q,i}^d)$, the flit under analysis does at most $h_b^i(x_{q,i}^d, y_{q,i}^d)$ hops through bypasses, all of which are consecutive links of the same column, and at most $(h_b^i(x_{q,i}^d, y_{q,i}^d) - 1)$ of those hops can be made by entering a router by port N . Furthermore, by Lemma 4, a high-priority flit cannot be deflected in two successive routers on the same column. Therefore, the flit under analysis may be deflected in at most half of the routers, i.e., in $\left\lceil \frac{h_b^i(x_{q,i}^d, y_{q,i}^d) - 1}{2} \right\rceil = \left\lfloor \frac{h_b^i(x_{q,i}^d, y_{q,i}^d)}{2} \right\rfloor$ routers, which proves the lemma. Note that, the last equality holds because $h_b^i(x_{q,i}^d, y_{q,i}^d) \in \mathbb{N}$. $\square$

The additional cost in terms of hops introduced by each deflection is analyzed in Lemma 6.

Lemma 6. The cost of a deflection is $c_{def} = S_x - 1$.

Proof. When a flit is deflected, it must hop through S_x routers on the ring to reach the same router as it would have if it could have used the bypass instead, i.e., through S_x routers instead of 1, thus leading to an additional cost of $S_x - 1$. $\square$

Injecting all the bounds proven in Lemmas 3, 5 and 6 into Equation (3.2), we can compute the WCTT of a flit of flow $f_{q,i}$.

3.4.2 Improved Analysis for the Worst-Case Traversal Time

The analysis proposed in Section 3.4.1 only uses information on the flit under analysis and does not rely on any information related to other communication flows that may be transmitted in the system. That analysis is thus useful for dynamic systems where the set of communication flows may vary over time. However, it may also be pessimistic and could be tightened if more information on the system is known. Indeed, the analysis always assumes that the flit of flow $f_{q,i}$ will suffer the maximum number of possible deflections. This may never happen in the real network if, for instance, there are no other flow using the same route than the flit of $f_{q,i}$. In this section, we derive more precise bounds on $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ based on the knowledge of the actual set of flows that can interfere with the transmission of a flit of flow $f_{q,i}$.

To derive the sets of flows that may interfere with a flit of flow $f_{q,i}$, we first define the sets $\Gamma_m^{N \rightarrow S}$, $\Gamma_m^{W \rightarrow S}$ and $\Gamma_m^{W \rightarrow E}$ as the sets of flows that traverse a router R_m from port N to port S , from port W to port S and from port W to port E , respectively, assuming that no deflection ever happens in the network.

Lemma 7. $\Gamma_m^{N \rightarrow S} = \{f_{p,j} \mid (x_{p,j}^d = x_m) \wedge h_b^j(x_m, y_m) > 0 \wedge h_b^j(x_{p,j}^d, y_{p,j}^d) \geq h_b^j(x_m, y_m)\}$.

Proof. According to HopliteRT*'s routing policy, a flow $f_{p,j}$ may request to leave by port S of R_m only if its destination is on the same column than R_m (i.e., $x_{p,j}^d = x_m$) and its destination is either R_m or further south than R_m (i.e., $h_b^j(x_{p,j}^d, y_{p,j}^d) \geq h_b^j(x_m, y_m)$). Further, a flow cannot enter by port N of R_m if it must not perform at least one hop on a bypass to reach R_m , i.e., we must have $h_b^j(x_m, y_m) > 0$. $\square$

Lemma 8. $\Gamma_m^{W \rightarrow S} = \{f_{p,j} \mid (x_{p,j}^d = x_m) \wedge h_b^j(x_m, y_m) = 0\}$

Proof. According to HopliteRT*'s routing policy, a *non-deflected* flow $f_{p,j}$ may enter by port W only if it does not need to use any bypass to reach R_m (i.e., $h_b^j(x_m, y_m) = 0$). Further, it may request leaving by port S of R_m only if its destination is on the same column than R_m (i.e., $x_{p,j}^d = x_m$). $\square$

Lemma 9. $\Gamma_m^{W \rightarrow E} = \{f_{p,j} \mid h_b^j(x_m, y_m) = 0 \wedge h_r^j(x_{p,j}^d, y_{p,j}^d) > h_r^j(x_m, y_m)\}$

Proof. According to HopliteRT*'s routing policy, a *non-deflected* flow $f_{p,j}$ may enter by port W only if it does not need to use any bypass to reach R_m (i.e., $h_b^j(x_m, y_m) = 0$). Further, it may request to exit by port E only if the number of hops it must do on the ring to reach its destination is larger than the number of hops it must do on the ring to reach R_m (i.e., $h_r^j(x_{p,j}^d, y_{p,j}^d) > h_r^j(x_m, y_m)$). $\square$

We define def_m^{hp} as a binary function that returns 1 if high-priority flows may be deflected in the router R_m .

Lemma 10. Let R_n be the router directly north to R_k , then

$$\text{def}_m^{hp} = \begin{cases} 1 & \Gamma_m^{N \rightarrow S} \cap \text{hp} \neq \emptyset \wedge (\Gamma_m^{W \rightarrow S} \cap \text{hp} \neq \emptyset \vee \text{def}_n^{hp} = 1) \\ 0 & \text{otherwise.} \end{cases} \quad (3.5)$$

Proof. According to HopliteRT*'s routing policy, a high-priority flow can be deflected to port E only if it is a flow coming from port N that conflicts for port S (i.e., there must be $\Gamma_m^{N \rightarrow S} \cap \text{hp} \neq \emptyset$). Furthermore, conflicting flows must be of high priority too, and must be entering by port W . Since the flows that were deflected in R_n are the only *deflected* flows that may enter by port W and request port S of R_m , there must either be $\text{def}_n^{hp} = 1$ or there is at least one *non-deflected* flow entering by port W and requesting port S (i.e., $\Gamma_m^{W \rightarrow S} \cap \text{hp} \neq \emptyset$). $\square$

Similarly, we define def_m^{lp} as a binary function that returns 1 if low-priority flows may be deflected in the router R_m .

Lemma 11. Let R_n be the router directly north to R_m , then

$$\text{def}_m^{lp} = \begin{cases} 1 & \Gamma_m^{N \rightarrow S} \cap \text{hp} \neq \emptyset \wedge (\Gamma_m^{W \rightarrow S} \cap \text{lp} \neq \emptyset \vee \text{def}_n^{lp} = 1) \\ 1 & \Gamma_m^{N \rightarrow S} \cap \text{lp} \neq \emptyset \wedge (\Gamma_m^{W \rightarrow S} \neq \emptyset \vee \text{def}_n^{lp} = 1 \vee \text{def}_n^{hp} = 1) \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

Proof. According to HopliteRT*'s routing policy, a low-priority flow can be deflected to port E under two possible scenarios: 1) it is a flow entering by port W that conflicts for port S (i.e., $\Gamma_m^{W \rightarrow S} \cap \text{lp} \neq \emptyset$) with a high priority flow coming from port N (i.e., $\Gamma_m^{N \rightarrow S} \cap \text{hp} \neq \emptyset$); or 2) if it is a flow coming from port N that conflicts for port S (i.e., $\Gamma_m^{N \rightarrow S} \cap \text{lp} \neq \emptyset$) with any flow coming from port W . We remind that the flows that were deflected in R_n are the only *deflected* flows that may enter by port W and request port S of R_m . Therefore, for scenario 2) to happen, there must be $\text{def}_n^{lp} = 1$ or $\text{def}_n^{hp} = 1$, or there is at least one *non-deflected* flow entering by port W and requesting port S (i.e., $\Gamma_m^{W \rightarrow S} \neq \emptyset$). Scenarios 1) and 2) directly correspond to the first and second case in Equation (3.6), respectively. $\square$

The functions def_k^{hp} and def_k^{lp} allow us to identify the set of routers in which a flit may be deflected. Therefore, thanks to them, we can now compute the maximum number of deflections a flit may suffer on its route to its destination. We first derive such bound for low-priority flits in Lemma 12.

Lemma 12. The maximum number of deflections suffered by a flit of a low priority flow $f_{q,i}$ with destination $(x_{q,i}^d, y_{q,i}^d)$ is

$$n_{def}^i(x_{q,i}^d, y_{q,i}^d) = \sum_{R_m \in \mathcal{R}^i} \text{def}_m^{lp} \quad (3.7)$$

where $\mathcal{R}^i = \{R_m \mid x_m = x_{q,i}^d \wedge y_m = (y_{q,i}^{o'} + j) \bmod S_y, \forall j = 0, \dots, h_b^i(x_{q,i}^d, y_{q,i}^d) - 1\}$.

Proof. The set $\mathcal{R}^i$ contains all the routers on the same column x than the destination of flow $f_{q,i}$ and that are on its route to its destination. That is, those routers between the Y-coordinate $y_{q,i}^{o'}$ (inclusive) and the destination $y_{q,i}^d$ of the flow. Therefore, it contains all the routers in which a flit of $f_{q,i}$ may be deflected (since a packet can only be deflected in a router on the same column of its destination and that is on its route to its destination). Since, by Lemma 11, def_m^{lp} returns 1 if a low priority flit may be deflected in router R_m , and 0 otherwise, the sum $\sum_{R_m \in \mathcal{R}^i} \text{def}_m^{lp}$ returns the total number of routers in which a packet of $f_{q,i}$ may be deflected. $\square$

The procedure to compute the maximum number of deflections suffered by a flit of a high-priority flow $f_{q,i}$ is a bit more complex. Let us first define the set $\mathcal{R}_{def}^i$ as the set of routers in which a high-priority flit may be deflected. That is, $\mathcal{R}_{def}^i = \{R_m \mid R_m \in \mathcal{R}^i \wedge \text{def}_m^{hp} = 1\}$ where $\mathcal{R}^i$ is defined as in Lemma 12. The size of that set is obviously an upper-bound on the number of deflections that a flit of $f_{q,i}$ may suffer. However, that value would be very pessimistic. Indeed, according to Lemma 4, the same high-priority flit cannot be deflected in two successive routers

in the same column. Lemma 13 integrates that information to compute a tighter bound on the maximum number of deflections suffered by a flit of $f_{q,i}$.

Lemma 13. Let $\mathbb{G}_{def}^i$ be a graph that contains one vertex per router in $\mathcal{R}_{def}^i = \mathcal{R}_{def}^i \setminus \{R_m | x_m = x_{q,i}^d \wedge y_m = y_{q,i}^{o'}\}$, and such that any two vertices V_i and V_j of $\mathbb{G}_{def}^i$ are connected by an edge if the routers R_i and R_j corresponding to those vertices are direct neighbors (i.e., they are connected by a $S \rightarrow N$ link). The maximum number of deflections suffered by a flit of a high-priority flow $f_{q,i}$ is the size of the maximum independent set of $\mathbb{G}_{def}^i$.

Proof. Let's first mention that the set $\mathcal{R}_{def}^i$ includes all routers located in the same column as the destination of flow $f_{q,i}$ along its route, where a high-priority flit may be deflected, excluding the first router it reaches with the same X-coordinate as its destination, i.e., $\mathcal{R}_{def}^i = \mathcal{R}_{def}^i \setminus \{R_m | x_m = x_{q,i}^d \wedge y_m = y_{q,i}^{o'}\}$. We remove this router from the set $\mathcal{R}_{def}^i$ since a flit of a high-priority flow will never experience deflection here, as only two scenarios can occur: (1) such router is indeed the origin router of the flit, then the flit is injected by the PE to its port S , and therefore there is no room for deflection, or (2) the flit arrives at that router after traveling along the X-axis, entering the router via port W and therefore always having the highest priority to use port S . Now, let's proceed to prove the lemma.

The maximum independent set of a graph $\mathbb{G}_{def}^i$ is the largest subset $\mathcal{S}$ of vertices of $\mathbb{G}_{def}^i$ such that any two vertices in $\mathcal{S}$ are not connected by an edge in $\mathbb{G}_{def}^i$.

Since vertices in $\mathbb{G}_{def}^i$ are connected by an edge if and only if they are neighbors in the NoC (i.e., they are connected by a $N \rightarrow S$ link), the maximum independent set $\mathcal{S}$ of $\mathbb{G}_{def}^i$ contains the largest possible number of routers from $\mathcal{R}_{def}^i$ that are not connected by a $S \rightarrow N$ link. That is, it contains the largest number of routers in which a flit of flow $f_{q,i}$ may be deflected and that are not successive routers in the same column of the network. Therefore, it contains the maximum number of routers in which a flit of $f_{q,i}$ may be deflected while respecting the constraint set by Lemma 4. $\square$

By means of an example, Figure 3.6 illustrates how Lemma 13 determines the maximum number of deflections suffered by a flit of a high-priority flow.

Example. Consider a high-priority red flit originating from router R_o and with destination router R_d (refer to Figure 3.6a). Let's assume that according to Equation 3.5, the flit could be deflected in routers at (1;0), (1;1), (1;2), (1;4), and (1;5). The router at coordinates (1;0) is the first router encountered by the flit, sharing the same X-coordinate as its destination router. In this router, the flit requests port S to start traveling along the Y-axis. However, it cannot be deflected because, after traveling along the X-axis, it reaches the router from port W and, therefore, always has the highest priority for using its port S . As a result, the set $\mathcal{R}_{def}^i$ only includes routers at coordinates (1;1), (1;2), (1;4), and (1;5). As stated in Lemma 4, if the flit is deflected in the router at (1;1), it cannot be deflected in the router at (1;2), and the same holds true for routers at (1;4) and (1;5). Then, Figure 3.6b shows the graph $\mathbb{G}_{def}$ for the red flit. Vertices V_1 , V_2 , V_4 , and V_5 correspond to routers at (1;1), (1;2), (1;4), and (1;5), respectively. Edges $e_{1,2}$ and $e_{4,5}$ connect pairs of vertices V_1 and V_2 , and V_4 and V_5 , respectively. Consequently, the maximum independent set of the graph includes

vertices V_1 and V_4 (see red circles in Figure 3.6b), resulting in a maximum of two deflections for the red flit in this example.

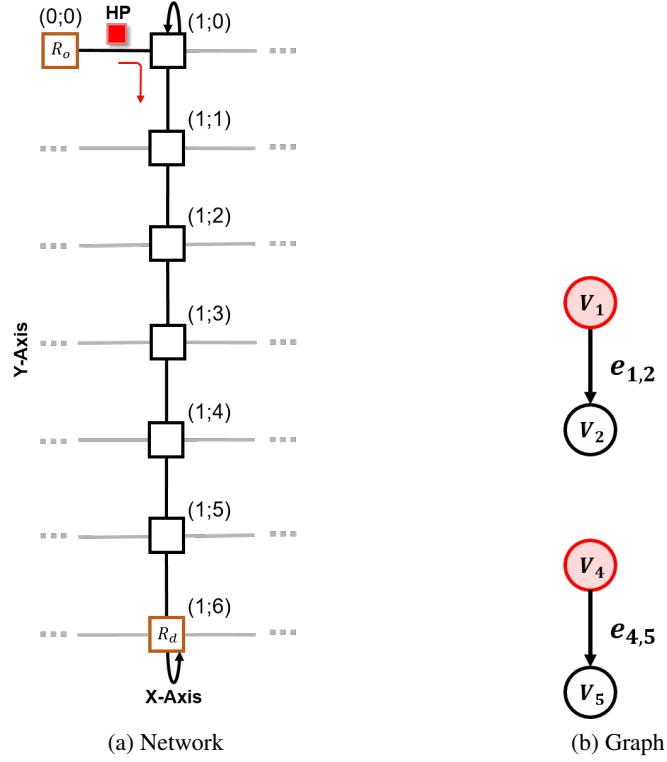


Figure 3.6: Example of how to calculate an upper-bound on the maximum number of deflections suffered by a flit of a high-priority flow.

The new bounds on the number of deflections provided in Lemmas 12 and 13 can then be used instead of those in Lemmas 3 and 5 to compute the WCTT with Equation (3.2).

3.4.3 Bound on the Worst-Case Injection Time

In the previous sections, we derived upper-bounds on the maximum traversal time of any flit of flow $f_{q,i}$. In this section, we provide an analysis of the worst-case injection time.

We first discuss the best-case injection scenario (Lemma 14) for flow $f_{q,i}$. The worst-case injection scenario is then discussed in Lemma 15.

Lemma 14. The flow $f_{q,i}$ can transmit at most $\lambda_i(t) = \min \left\{ t, \left\lceil \frac{t + wcit_i}{T_{q,i}} \right\rceil C_{q,i} \right\}$ flits in any time interval of length t .

Proof. Since at most one flit can be sent by $f_{q,i}$ every clock cycle, it holds that

$$\lambda_i(t) \leq t. \quad (3.8)$$

Moreover, let $wcit_i$ be the WCIT suffered by any packet of flow $f_{q,i}$. Then, $f_{q,i}$ injects the most flits in an interval of length t when one of its packets was kept from being injected during $wcit_i$

cycles before the beginning of the interval, that packet starts to be injected right at the start of the interval, new packets are generated with their minimum inter-arrival time $T_{q,i}$ and no flit of $f_{q,i}$ suffers blocking during the interval of length t . Under such conditions, we have

$$\lambda_i(t) \leq \left\lceil \frac{t + wcit_i}{T_{q,i}} \right\rceil C_{q,i}. \quad (3.9)$$

Since the minimum of two upper-bounds is an upper-bound, the minimum of (3.8) and (3.9) is an upper-bound on $\lambda_i(t)$. $\square$

Now that we discussed the best-case scenario, we consider the worst-case delay that a flow may experience to inject a packet in the network. To ease the discussion, we denote by Γ_p^C the set of flows that conflict with the injection of a flit of $f_{q,i}$ at router R_q . Let's assume that the set of conflicting flows Γ_q^C is known. We can upper-bound $wcit_i$ as in Lemma 15.

Lemma 15. Let l be a flit of flow $f_{q,i}$ injected at router R_q with coordinates (x_q, y_q) . The WCIT $wcit_i$ caused by flows conflicting with flit l is given by the smallest positive solution to the recursive equation

$$wcit_i \geq \left(\sum_{\forall f_{q,j} \in \mathcal{J}_q} C_{q,j} \right) - 2 + \sum_{\forall f_{p,k} \in \Gamma_q^C} \lambda_k(wcit_i + J_k + 1) \quad (3.10)$$

where

$$J_k = n_{def}^k(x_q, y_q) \times c_{def} \quad (3.11)$$

and

$$\mathcal{J}_q = \begin{cases} F_q & \text{if } P_{q,i} = \text{low}; \\ F_q \cap \text{hp} & \text{if } P_{q,i} = \text{high}. \end{cases} \quad (3.12)$$

Proof. According to HopliteRT*'s routing policy, the PE π_q will be able to inject the last flit of a packet of flow $f_{q,i}$ as soon as, (1) all flits previously pending in the FIFO queues of π_q have been injected, and (2) there is one clock cycle where no flit coming from other PE conflicts for the same output port than l . This happens as soon as the length of the interval is larger than the maximum number of pending flits that must be injected by π_q and the maximum number of flits generated by conflicting flows injected by other PEs that conflict to access the same output port during the interval.

The term referred to in point (1) is given by the maximum number of flits that may be generated by flows on π_q that are ahead of l in the FIFO queues of π_q . Since each flow may have at most one packet in the FIFO queue at a time (see Section 3.2.3), we have that the maximum number of flits ahead of l is strictly smaller than $\sum_{\forall f_{q,j} \in F_q} C_{q,j}$. Furthermore, according to Section 3.2.3, low priority flows injected by π_q cannot block high-priority ones. Therefore, if the priority $P_{q,i}$ of $f_{q,i}$ is high, then strictly less than $\sum_{\forall f_{q,j} \in F_q \cap \text{hp}} C_{q,j}$ flits may be sent ahead of l in π_q . Summarizing, at most

$$\left(\sum_{\forall f_{q,j} \in \mathcal{J}_q} C_{q,j} \right) - 1 \quad (3.13)$$

flits generated by π_q (with $\mathcal{J}_q$ defined as in the claim) may interfere with the injection of l .

To compute the term referred by point (2) in the explanation above, consider the flow $f_{p,k}$ that may conflict with the packet p under analysis (i.e., $f_{p,k} \in \Gamma_q^C$). By definition of $n_{hops}^k(x_q, y_q)$, flits from $f_{p,k}$ will reach the router (x_q, y_q) in no less than $n_{hops}^k(x_q, y_q)$ clock cycles. That is, the *last flit* generated by $f_{p,k}$ that may conflict with the injection of l must have been injected into the NoC *no later* than $n_{hops}^k(x_q, y_q)$ clock cycles *before the end* of the period during which l is interfered with. Similarly, according to Equation (3.2), flits from flow $f_{p,k}$ will reach the router (x_q, y_q) in no more than $(n_{hops}^k(x_q, y_q) + n_{def}^k(x_q, y_q) \times c_{def})$ clock cycles. Thus, the *first flit* generated by $f_{p,k}$ that may conflict with the injection of l must have been injected *no earlier* than $n_{hops}^k(x_q, y_q) + n_{def}^k(x_q, y_q) \times c_{def}$ clock cycles *before the start* of the interference with l . Therefore, the length of the interval during which $f_{p,k}$ may inject flits that conflict with the flit l under analysis is given by $\Delta t = wcit_i + 1 - n_{hops}^k(x_q, y_q) + n_{hops}^k(x_q, y_q) + n_{def}^k(x_q, y_q) \times c_{def} = wcit_i + J_k$ where $J_k = n_{def}^k(x_q, y_q) \times c_{def}$ and $(wcit_i + 1)$ is the duration of the time interval starting when the packet to which l belongs is inserted in R_q 's FIFO queue and finishing when the flit l is injected into the network.

As proven in Lemma 14, a flow $f_{p,k}$ can inject at most $\lambda_k(\Delta t)$ flits into the network in any time interval of length Δt . Therefore, all the flows that may interfere with the injection of l can inject at most

$$\sum_{\forall f_{p,k} \in \Gamma_q^C} \lambda_k(wcit_i + J_k + 1) \quad (3.14)$$

flits that may conflict with l .

Combining Equations (3.13) and (3.14), we get that l may be injected as soon as

$$wcit_i + 1 \geq \left(\sum_{\forall f_{q,j} \in \mathcal{J}_q} C_{q,j} \right) - 1 + \sum_{\forall f_{p,k} \in \Gamma_q^C} \lambda_k(wcit_i + J_k + 1)$$

Hence proving the lemma. $\square$

Lemma 16. The set of flows Γ_q^C conflicting with the injection of a flit of flow $f_{q,i}$ at router R_q is defined as

$$\Gamma_q^C = \Gamma_q^S \cup \Gamma_q^E \quad (3.15)$$

where Γ_q^S and Γ_q^E are the set of flows that may request port S and E of R_q , respectively.

Proof. According to HopliteRT*'s routing policy (see cases 3, 4, and 5 in Table 3.1), PEs always have the lowest priority to use output ports. Consequently, a PE can only inject a new flit into the network once the desired output port becomes available. Specifically, a PE is prevented from injecting a flit to port S of R_q if there is a flow entering the router from port W or N requesting port S . However, a PE is prevented from injecting a flit to port E of R_q under the following conditions: (1) if there is a flow entering the router from port W and requesting port E (see case 6 in Table 2.1); (2) if two flows enter the router simultaneously and request port S at the same time, as one of the flows is deflected to port E (see case 2 in Table 2.1); or (3) if there is a flow entering the router

from port W and requesting port S , since according to HopliteRT*'s routing policy (see case 8 in Table 3.1), the PE is unable to inject flits into the network under this scenario. This implies that the injection of a flit can be prevented by any communication flow entering the router and requesting port S or E . Therefore, the set of flows Γ_q^C conflicting with the injection of a flit of flow $f_{q,i}$ at router R_q is composed of all the flows injected by other PEs that may request port S or E of R_q . In other words, this set includes all the flows belonging to the sets Γ_q^S and Γ_q^E , respectively. Thus, the lemma is proven. $\square$

We define Γ_q^S and Γ_q^E in Lemmas 19 and 20. However, to compute Γ_q^S and Γ_q^E , we must first define the set Γ_q^{def} of flows that may be deflected in router R_q . We further divide that set in $\Gamma_q^{\text{def_hp}}$ and $\Gamma_q^{\text{def_lp}}$ such that $\Gamma_q^{\text{def_hp}}$ is the set of high priority flows that may be deflected in R_q , and $\Gamma_q^{\text{def_lp}}$ is the set of low priority flows that may be deflected in R_q . By definition, $\Gamma_q^{\text{def}} = \Gamma_q^{\text{def_hp}} \cup \Gamma_q^{\text{def_lp}}$.

Lemma 17.

$$\Gamma_q^{\text{def_hp}} = \begin{cases} \Gamma_q^{N \rightarrow S} \cap hp & \text{if } \text{def}_q^{hp} = 1 \\ \emptyset & \text{otherwise.} \end{cases}$$

Proof. According to HopliteRT*'s routing policy, only high-priority flows entering by port N and contending for port S can be deflected (i.e., all flows in $\Gamma_q^{N \rightarrow S} \cap hp$). Therefore, $\Gamma_q^{\text{def_hp}} = \Gamma_q^{N \rightarrow S} \cap hp$ when deflections may happen in router R_q (i.e., when $\text{def}_q^{hp} = 1$). If no deflection may happen in R_q (i.e., $\text{def}_q^{hp} = 0$) then the set of deflected flows in R_q is obviously empty. $\square$

Lemma 18.

$$\Gamma_q^{\text{def_lp}} = \begin{cases} \{\Gamma_q^{W \rightarrow S} \cup \Gamma_q^{N \rightarrow S} \cup \Gamma_n^{\text{def}}\} \cap lp & \text{if } \text{def}_q^{lp} = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

with R_n being the router directly north to R_q .

Proof. First, remember that according to HopliteRT*'s routing policy, any flow that is deflected in R_n (i.e., flows in Γ_n^{def}) will enter in R_q by port W and compete for port S . Further, any low priority flow entering by port N or port W and contending for port S can be deflected (i.e., all flows in $\{\Gamma_q^{W \rightarrow S} \cup \Gamma_q^{N \rightarrow S} \cup \Gamma_n^{\text{def}}\} \cap lp$). Therefore, $\Gamma_q^{\text{def_lp}} = \{\Gamma_q^{W \rightarrow S} \cup \Gamma_q^{N \rightarrow S} \cup \Gamma_n^{\text{def}}\} \cap lp$ when deflections may happen in router R_q (i.e., when $\text{def}_q^{lp} = 1$). If no deflection may happen in R_q (i.e., $\text{def}_q^{lp} = 0$) then the set of deflected flows in R_q is obviously empty. $\square$

Now, we can define Γ_q^S and Γ_q^E .

Lemma 19. The set of flows coming from other routers that conflict on port S of router R_q is given by $\Gamma_q^S = \Gamma_q^{N \rightarrow S} \cup \Gamma_q^{W \rightarrow S} \cup \Gamma_n^{\text{def}}$ where R_n is the router directly north to R_q .

Proof. The port S of router R_q may be kept busy by any flow entering by port N or W of R_q and requesting port S . Since according to HopliteRT*'s routing policy, the only deflected flows that may enter by port W of R_q and request port S are those deflected in R_n (i.e., flows in Γ_n^{def}), the set of all flows that may request port S of R_q is $\Gamma_q^{N \rightarrow S} \cup \Gamma_q^{W \rightarrow S} \cup \Gamma_n^{\text{def}}$ (where $\Gamma_q^{N \rightarrow S} \cup \Gamma_q^{W \rightarrow S}$ is the set of all flows that were not deflected that may enter R_q and request port S). $\square$

Lemma 20. The set of flows coming from other routers that conflict on port E of router R_q is given by $\Gamma_q^E = \Gamma_q^{W \rightarrow E} \cup \Gamma_{ring}^{def} \cup \Gamma_q^{W \rightarrow S} \cup \Gamma_n^{def}$ where R_n is the router directly north to R_q and

$$\Gamma_{ring}^{def} = \bigcup_{\forall R_m: (x_m > x_q \wedge y_m = y_q - 1) \vee (x_m < x_q \wedge y_m = y_q)} \{\Gamma_m^{def}\}. \quad (3.16)$$

Proof. Let R_n be the router that is directly north to R_q (i.e., router at coordinates $(x_q, y_q - 1)$). Then according to HopliteRT*'s routing policy, the only deflected flows that may enter by port W of R_q and request port E are those that were deflected in routers located between R_n and R_q on the ring of the NoC (i.e., any router R_m such that $(x_m > x_q \wedge y_m = y_q - 1) \vee (x_m < x_q \wedge y_m = y_q)$). The set of all those flows is provided by Γ_{ring}^{def} . Therefore, the port E of R_q may only be requested by the flows in $\Gamma_{ring}^{def} \cup \Gamma_q^{W \rightarrow E}$, where $\Gamma_q^{W \rightarrow E}$ is the set of flows that are not deflected in any router and request port E of R_q (note that there is no flow that may enter by port N and request port E). Finally, the routing table of HopliteRT* (see Table 3.1) does not allow a flit of flow $f_{q,i}$ to be injected by the PE towards port E whenever there is flits entering R_q by port W and requesting port S . Thus, the set of all flows entering by port W and requesting port S (i.e., $\Gamma_q^{W \rightarrow S} \cup \Gamma_n^{def}$ as proven in Lemma 19) must be added to the set of conflicting flows, proving the lemma. $\square$

The results of Lemmas 19 and 20 can then be used to compute the WCIT using Lemma 15 and Equation (3.15).

3.5 Experimental Results

3.5.1 Implementation of HopliteRT*

We implemented HopliteRT* with the hardware description language Verilog taking advantage of the possibility of fracturing the look-up tables (LUTs) in recent FPGA platforms (i.e., the possibility to use a single LUT to implement two functions that would normally require two different LUTs) to reduce the implementation cost of expensive multiplexers. Specifically, in HopliteRT*, routers are implemented using two multiplexers of three inputs (see Figure 3.4), where each 3:1 multiplexer can be implemented with a 5-inputs LUT. Since the modern families of Xilinx FPGAs present 6-inputs LUTs that can be fractured into two 5-inputs LUTs sharing the same five input signals, the two multiplexers of a HopliteRT* router can be implemented with a single 6-inputs LUT as they share the same input signals.

We synthesized a single router of HopliteRT* for flits of 64 bits. The target platform was a Xilinx Kintex-7 FPGA. Kintex-7 is a mid-range FPGA family that contains between 65,600 and 477,760 LUTs, depending on the device selected. A router of HopliteRT* required 88 LUTs and 139 Flip-Flops (FFs), i.e., between 0.02% 0.13% of the total number of LUTs available in a Kintex-7 FPGA.

We compared HopliteRT* with HopliteRT [90] and three VC-based NoCs: CONNECT [67], IDAMC [88], and ProNoC [59] in terms of hardware resource utilization (see Table 3.2). A single

router of HopliteRT requires 86 LUTs. That is, the HopliteRT* router is only 2 LUTs (after fractionation) more expensive in comparison to HopliteRT. According to [67] and [88], a single router of CONNECT and IDAMC NoC requires about 1500 and 1300 LUTs, respectively. Thus, an 8x8 CONNECT NoC requires ≈ 96000 LUTs, i.e., between 20% and 147% of Kintex-7 resources, while an 8x8 IDAMC NoC requires ≈ 83000 LUTs, i.e., between 18% and 127%. A ProNoC router with two VCs requires 1574 LUTs. An 8x8 ProNoC NoC then requires ≈ 100000 LUTs, i.e. between 20% and 150% of the total number of LUTs available in a Kintex-7 FPGA. Therefore, as reported in Table 3.2, to implement an 8x8 VC-based NoC are needed between 80000 and 100000 LUTs. Then, a Kintex-7 FPGA does not provide enough hardware resources to embed these expensive NoC designs along with a large number of PEs. These solutions are not suitable for systems implemented on mid-range platforms. Conversely, an 8x8 HopliRT* NoC requires 5632 LUTs, i.e., between 1.1% and 8.5% of the resources. Thus, it is approximately 15-times cheaper than VCs-based NoCs in the studied configurations.

Table 3.2: Resources utilization of different NoCs in Kintex-7 FPGAs.

NoC	LUTs	% Resource utilization of the platform
8x8 HopliteRT	5504	1.1%-8.4%
8x8 CONNECT	96000	20%-147%
8x8 IDAMC	83000	18%-127%
8x8 ProNoC	100000	20%-150%
8x8 HopliteRT*	5632	1.1%-8.5%

Finally, we connected each router of a 4x4 HopliteRT* NoC to a Microblaze soft-core and synthesized the solution for a Virtex-7 485T FPGA. We computed the maximum operating frequency of the network with Xilinx Vivado. We obtained ≈ 275 MHz for both HopliteRT and HopliteRT*, thereby showing no degradation when adopting HopliteRT*.

3.5.2 NoC Latency Bounds for randomly generated flows

In this section, we provide experimental results by computing the WCTT, WCIT, and WCCT of sets of communication flows generated under different system configurations, (i.e., distinct NoC sizes, number of flows, and traffic patterns).

We generated sets of flows according to two traffic patterns, random and all2one.

- ▷ *random*: the origin and destination coordinates are randomly generated using a uniform probability distribution of a flow to originate and target any router in the network.
- ▷ *all2one*: origins are randomly generated but the same destination coordinates are assigned to all the flows. A priority level (low or high) is randomly assigned to each flow. The number of high-priority flows was roughly kept at 50% of the total number of flows in the network. The number of flits of packets released by a communication flow was randomly chosen between 1 and 5, and their inter-arrival times were generated as in [82].

In Figure 3.7, we provide the results computed by using the analysis of HopliteRT [90, 91], the analysis presented in Section 3.4.1, and the improved analysis of Section 3.4.2. We show the evolution of the maximum and average packets WCTT for an increasing number of flows in a 16x16 network considering a random and all2one traffic pattern. Each point in the plot is the result of 100 experiments. We varied the number of generated flows from 10 to 300 by steps of 10.

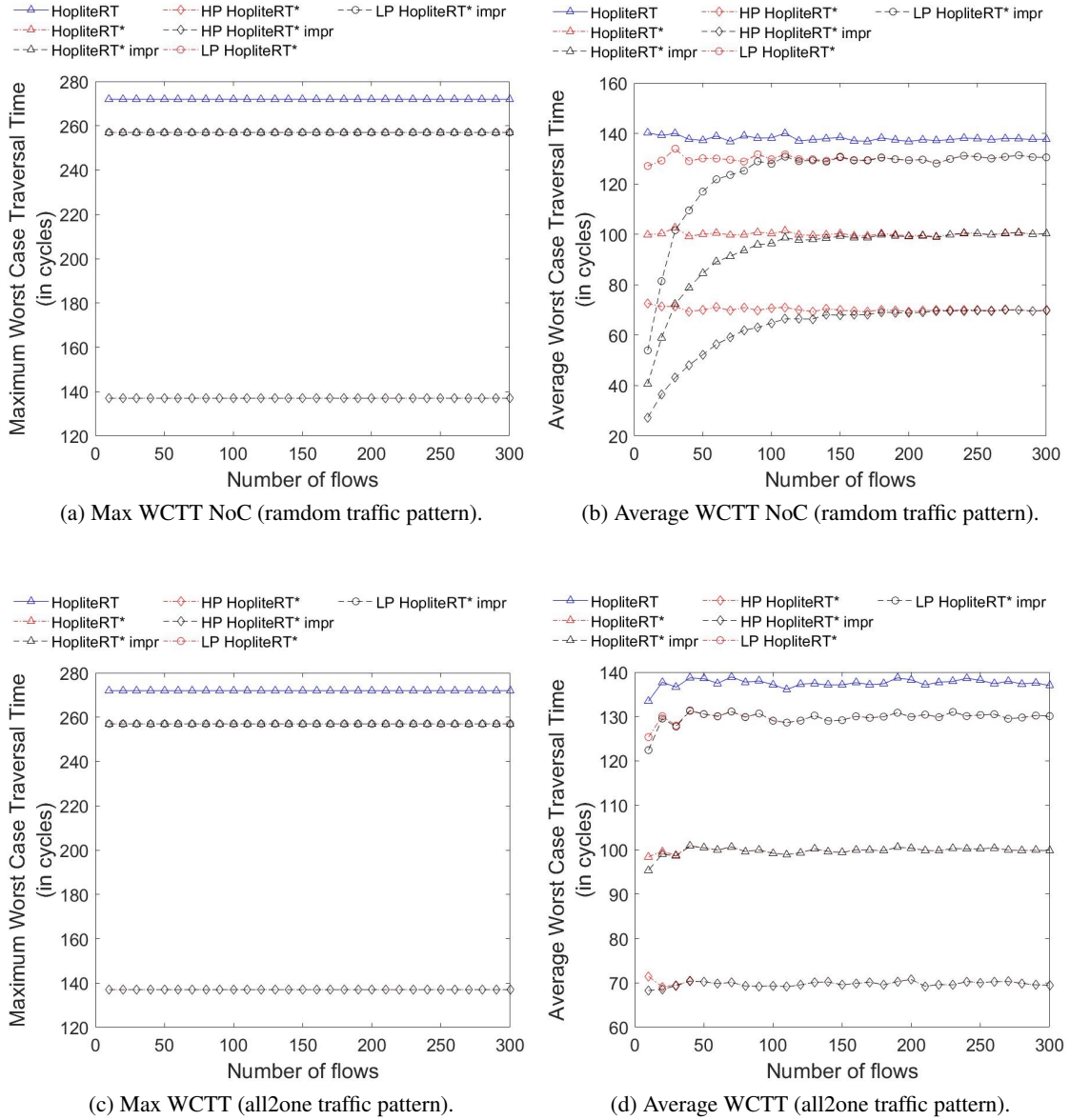


Figure 3.7: Experimental results for 16x16 NoC.

The maximum WCTT observed overall flows for both HopliteRT and HopliteRT* is roughly the same. This can easily be explained by the fact that HopliteRT* has the same WCTT bound as HopliteRT for low-priority flows. However, high-priority packets see their WCTT drastically reduced in HopliteRT*. It is even more visible when looking at the results returned by the improved

WCTT analysis, which reduces considerably the analysis pessimism by considering the actual set of flows that may interfere with each other. We also see that the improved analysis gain reduces as the number of flows increases. This is expected since the number of interfering flows and hence the number of deflections that flows may suffer increases when more packets are injected into the network. Even though, we show that the average WCTTs of high-priority flows remain considerably lower than that of other flows. We also show that the modifications of HopliteRT introduced in this paper allow at least a 2x improvement on the worst and average case traversal time of high-priority packets, without impacting the quality of service of low-priority packets, and up to 5x improvement on the average traversal time when there are few flows. With a `all2one` traffic pattern, the improved WCTT bound of HopliteRT* returns the same result as the simple bound of Section 3.4.1. That can be explained by the fact that most packets compete for the same resources (i.e., links and routers output ports) since their destination is the same. Hence, the number of conflicting flows and the number of deflections increase considerably for all the packets in comparison to the random traffic pattern.

In Figures 3.8a and 3.8b, we show the maximum and average WCCT of a set of flows with origin and destination routers randomly chosen in a 4x4 NoC. Those results were obtained by using the improved analysis of HopliteRT* and that presented in [53] by Liu et al., which is an improved analysis of that proposed in [82, 81] by Shi and Burn.

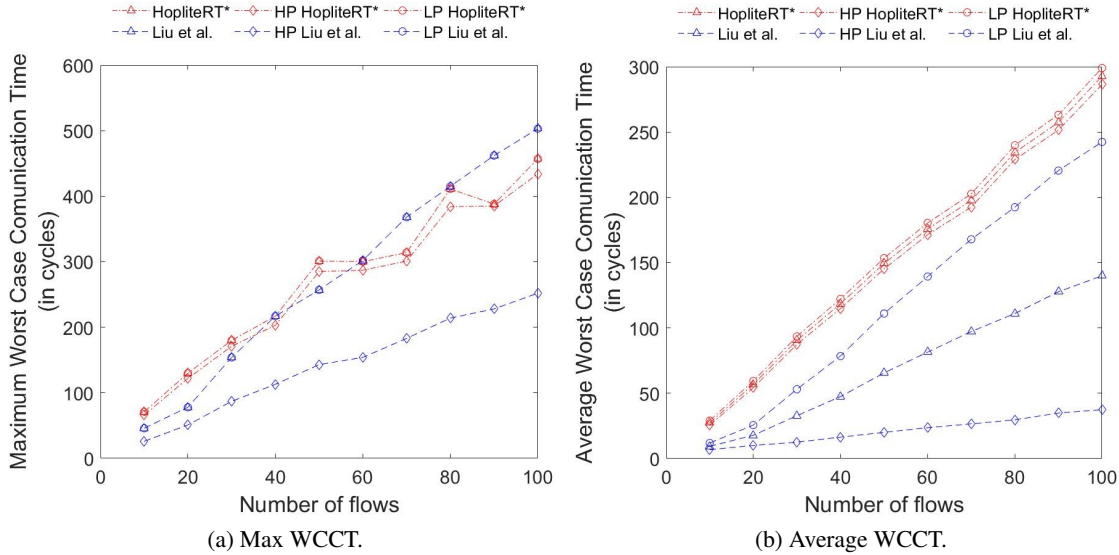


Figure 3.8: Experimental results for 4x4 NoC and random traffic pattern.

To establish a fair comparison, we assume two VCs (i.e., two priority levels) for the analysis presented in [53]. We observe that the analysis by Liu et al. performs better than that of HopliteRT* in terms of average and maximum worst-case communication time in most cases. We assume that it happens because [53] considers that each flow can only have one packet traversing the network at the same time, while HopliteRT* supports the transmission of multiple packets from

the same flow simultaneously, leading to more possible contentions, as well as, more pessimism in the HopliteRT*'s WCIT analysis. However, we recall that a router similar to that assumed by [53] is likely 10 to 20 times more costly from a hardware viewpoint than a HopliteRT* router (see Table 3.2).

3.5.3 Use-case analysis

We used an autonomous vehicle application presented in [53, 79] to compare the improved analysis of HopliteRT* and that by Liu et al. The application is composed by multiple tasks that rely on 38 traffic flows to communicate. It is mapped on 16 PE connected to a 4x4 NoC. The parameters of traffic flows and their origin and destination routers are kept the same as those used in [79]. The priority was assigned according to their period and deadline, that is, flows f_8 to f_{30} were given the highest priority since their periods/deadlines are the shortest. In Figure 3.9, we show that the WCCT of 14 flows is noticeably better with HopliteRT* in comparison to Liu et al.'s analysis [53]. The WCCT of 12 flows is noticeably better with Liu et al.'s analysis, and the last 12 flows have comparable results with both analyses. Therefore, we conclude that in a real use-case, there is no analysis that dominates the other.

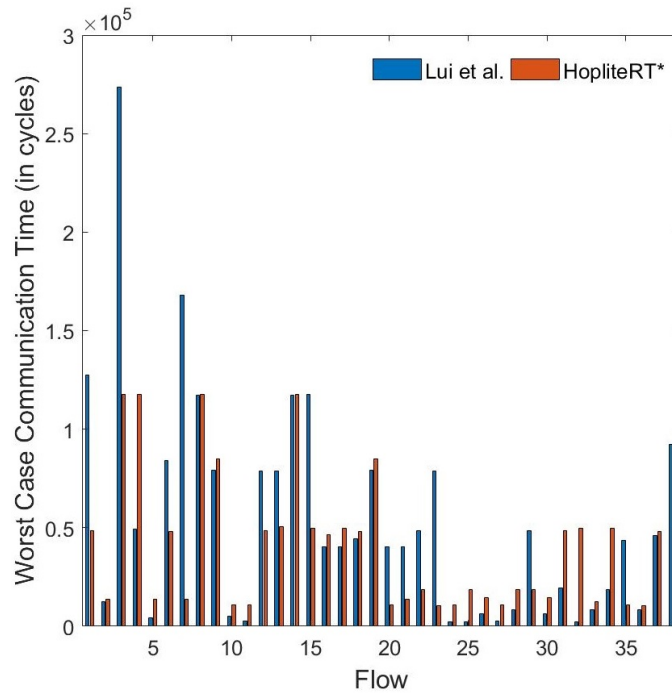


Figure 3.9: Experimental results for the case study.

3.5.4 RTL Simulations

We performed cycle-accurate simulations of HopliteRT and HopliteRT* using HDL Verilog implementations of 4x4 NoCs. We configured each PE to inject one, two, or three flows in the network. We generated flows with a random traffic pattern and random priorities. The inter-arrival

times of flows were randomly chosen within the set $\{100, 200, \dots, 1000\}$. The utilization of each PE was set at 20%, and then the utilization of each flow from the same PE was generated using [8]. The number of flits in a flow is given by multiplying its utilization by its inter-arrival time. The flit size was set to 64 bits.

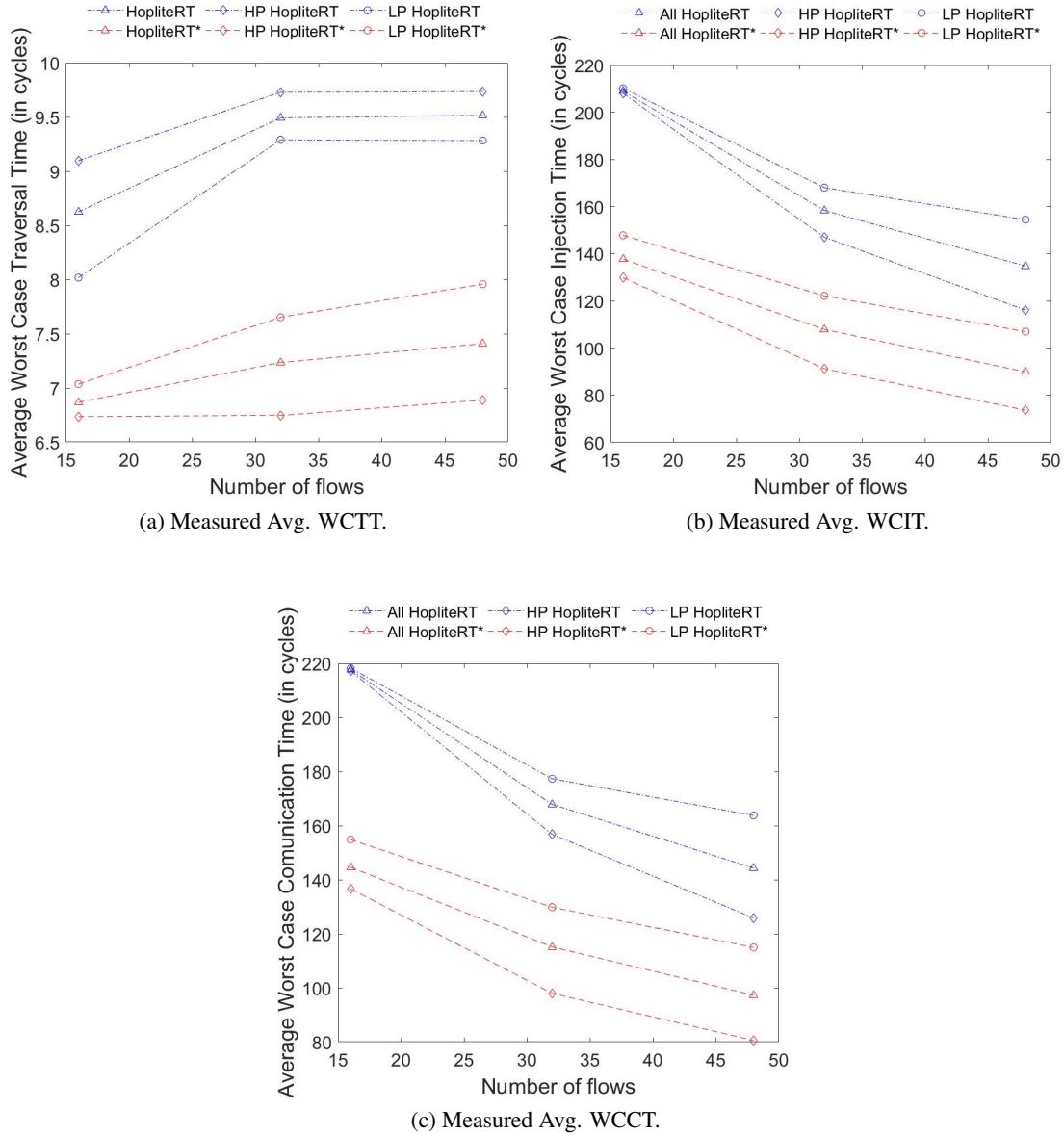


Figure 3.10: Experimental results for 4x4 NoC and random traffic pattern.

In Figure 3.10a, we provide the *measured* average WCTT for HopliteRT and HopliteRT*. We observe that packets reach their destination considerably faster with HopliteRT*. Note that, the WCTT of high-priority packets is higher than that measured for low-priority ones in HopliteRT. That is due to HopliteRT not making any distinction between high and low flows. However, we show that by using HopliteRT* the quality of service is guaranteed to flows of high prior-

ity and hence their WCTT decreases. We provide the measured average WCIT and WCCT in Figures 3.10b and 3.10c, respectively.

In Figure 3.11, we present the measured average WCTT in HopliteRT* for a set of 32 flows against the improved version of the bound introduced in this paper. In this experiment, each PE can inject packets from two different flows in the network. We observe that our approach provides safe and mostly tight upper-bounds on the WCTT for high and low priority flows.

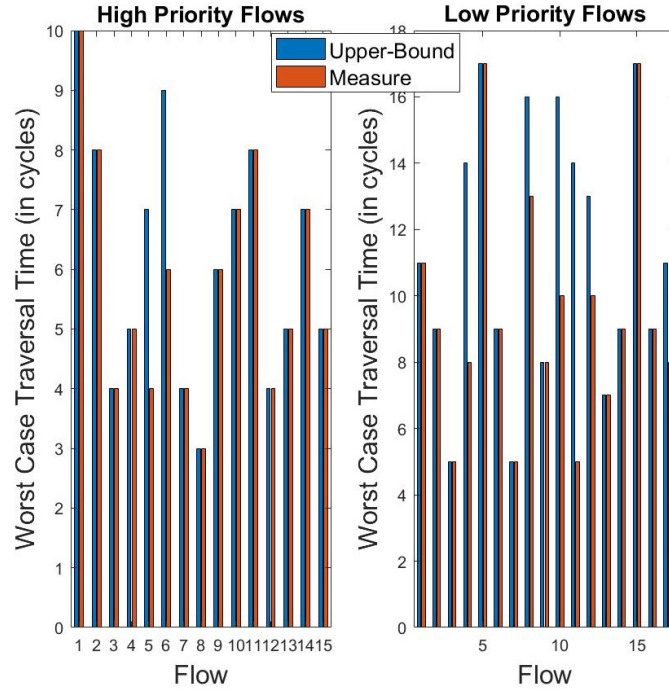


Figure 3.11: WCTT of randomly generated packets traversing a 4x4 NoC.

3.6 Chapter Summary

The chapter introduced HopliteRT*, a novel NoC design that incorporates a priority-based routing policy and significantly improves timing performance while only marginally increasing hardware resource utilization compared to HopliteRT. HopliteRT* offers several appealing features, including being buffer-less, predictable, and easy to analyze in terms of timing behaviors. It also demonstrates adaptability to dynamic workloads and suitability for low-power and resource-constrained systems. By adopting the circulant topology, HopliteRT* reduces the number of deflections, resulting in lower WCTT for high priority packets. Additionally, we identified an issue in the timing analysis of HopliteRT and proposed a new timing analysis for HopliteRT*. The chapter provided a complete implementation of HopliteRT* in HDL Verilog, enabling cycle-accurate emulation on a Xilinx FPGA. Through synthetic experiments and comparison with related works, we demonstrated substantial performance improvements, confirming the efficacy of HopliteRT* as a promising NoC solution for real-time applications.

Chapter 4

Ensuring In-Order Flit Delivery at the Destination

In HopliteRT*, the number of deflections suffered by the flits, and thus the length of their routes, depends on the level of congestion they encounter during their journey towards their destination. Consequently, flits from the same packet that have been sequentially injected into the network may follow different routes while traveling to the same destination. Due to the nondeterministic number of deflections that flits may experience while traversing the network, they may not reach their destination in the same order as they were injected at their sources. In other words, they may arrive at the destination in an out-of-order fashion. Follows, an example illustrating this behavior.

Example. Consider a 4x4 HopliteRT* NoC (see Figure 4.1). Also, consider a low-priority packet p split into 3 flits sequentially injected at router R_o with destination router R_d (red packet in Figure 4.1). Flit 1 of p enters router R_m via port N . Then, assume that it contends over port S of router R_m with a high-priority green flit coming from port W (see Figure 4.1a). According to HopliteRT*'s routing policy (row 2 Table 3.1), the green flit wins access to port S , and flit 1 of p is deflected to port E . In the next cycle (see Figure 4.1b), flit 2 of p is routed $N \rightarrow S$ at router R_m , as it does not conflict with other flits for port S . One cycle later, flit 2 is the first one from p to reach R_d (see Figure 4.1c). Therefore, flits reach the destination in an out-of-order fashion.

In this chapter, we propose IPDeN, a deflection-based NoC that ensures in-order flit delivery at the destination and simplifies packet reconstruction. IPDeN keeps some of the most compelling design features of HopliteRT*, i.e., the network topology, which ensures that flits always progress toward their destination even when deflected, and the intuition behind the routing policy, which is easy to understand, cost-effective to implement, and ensures predictable timing behavior. With IPDeN, we propose a new router architecture associated to a new routing policy that guarantees flits arrive at their destination in the same order as they were injected at their source. To void the use of costly reordering logic at the destination while preventing flits from overtaking other flits belonging to the same communication flow, IPDeN incorporates small buffers into its design. However, as we will discuss in this section, the size of these buffers is solely dependent on the network topology, thereby distinguishing it from SotA solutions.

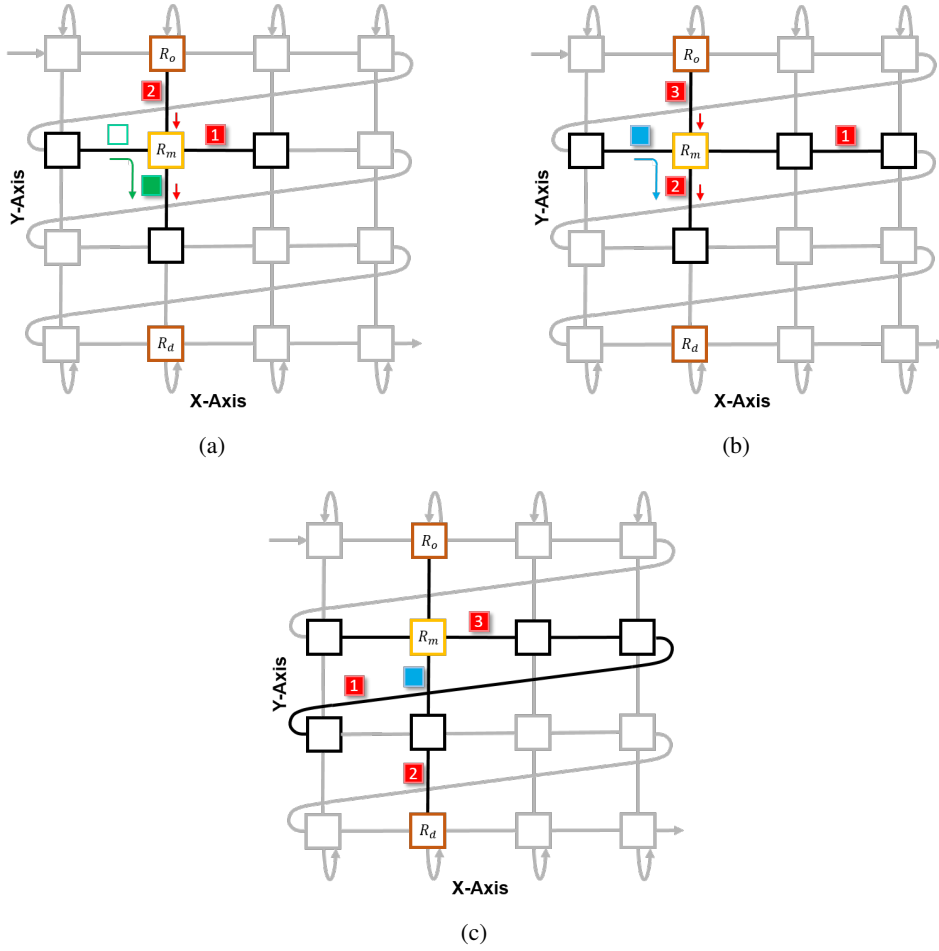


Figure 4.1: Example of an out-of-order reception in HopliteRT*.

The chapter is structured as follows. The system model is updated (in comparison to the model assumed in Chapter 3) in Section 4.1. Next, we present and discuss all the design features of IPDeN in Sections 4.2 and 4.3. Section 4.4 is dedicated to derive a WCTT analysis for packets transmitted over IPDeN and in Section 4.5 we discuss network latency. Finally, in Section 4.6, experimental results are presented.

The content of this chapter was previously published in the following paper:

- **Y. Ribot González**, G. Nelissen and E. Tovar, IPDeN: Real-Time deflection-based NoC with in-order flits delivery. In 28th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'22), pp. 160-169, 2022, doi:10.1109/RTCSA55878.2022.00023.

4.1 System Model

The system model in IPDeN is similar to that of HopliteRT* (refer to Section 3.1). However, unlike HopliteRT*, IPDeN does not consider packets' priority. We elaborate on the main reason

behind this design choice in Section 4.3.3. Hence, each flow $f_{q,i}$ is now defined by the 4-tuple $\langle C_{q,i}, T_{q,i}, R_{q,i}^o, R_{q,i}^d \rangle$.

4.2 IPDeN Design

4.2.1 Routing Policy

IPDeN maintains the same network topology and nearly identical router interface (i.e., input and output ports) as HopliteRT*. That is, in IPDeN, the routers are connected together according to the ring circulant topology $(N; g_1, g_2)$ presented in Section 3.2.2 (see Figure 3.3). Under this topology, each router of IPDeN has four inputs and four output ports. Two input ports referred to by the letters N and W (for North and West), and two output ports referred to as E and S (for East and South) are used for inter-routers communication. The remaining two input ports, named PEi_1 and PEi_2 , are used by the PE to inject packets on ports E and S , respectively. When a flit reaches its destination router, it is transmitted to the PE via either of the remaining two output ports named PEo_1 and PEo_2 . To reduce the injection time of packets, we assume that packets may be injected via PEi_1 and PEi_2 in parallel. Therefore, a packet that is waiting to be injected into the network only conflicts with the subset of packets that must be injected to the same input port PEi_u .

Specifically for IPDeN, we assume the following property regarding which port is used to inject flits into the network.

Property 1. In IPDeN, we assume that a flit of a flow $f_{q,i}$ with origin coordinates $(x_{q,i}^o, y_{q,i}^o)$ and destination coordinates $(x_{q,i}^d, y_{q,i}^d)$, is injected in the network using port PEi_1 **if and only if** $x_{q,i}^o \neq x_{q,i}^d$ and using port PEi_2 **if and only if** $x_{q,i}^o = x_{q,i}^d$.

From Property 1, we get that all packets released by programming elements π_q and belonging to the same communication flow will be injected in the network using the same input port of R_q .

IPDeN's routing policy is mostly similar to that of HopliteRT [90]. A flit of flow $f_{q,i}$ would first travel horizontally along the X-axis of the network until it reaches a router with the same X coordinate as its destination router. Once in that router, it requests the S port and travels vertically along the Y-axis until reaching its destination. The main difference between HopliteRT and IPDeN is related to the arbitration between flits injected by the PE and flits entering by the ports W and N of the router. IPDeN's routing policy is summarized in Table 4.1. The text highlighted in red indicates the variations of IPDeN's routing policy in comparison to HopliteRT.

In IPDeN, flits entering by the port W may request the ports E or S , while flits entering by the port N may only request the port S . Flits entering by the port W and requesting E never suffer contention, i.e., they travel freely through the X-axis (row 1 in Table 4.1). In addition, a flit entering a router by the port W always has the highest priority to use the port S when it conflicts with a flit entering by the port N . Under this scenario, the flit coming from the port N is deflected to the port E inconsiderately of its final destination (row 2 in Table 4.1). The deflected flit will now travel along the X-axis. It will enter the next routers via the port W and will thus have the highest priority the next time it will require an S port. Finally, flits entering by the ports PEi_1 and

thereby pointing to the last slot of the buffer. At each clock cycle, if a flit is routed to port S , it is saved at the buffer position pointed by B_q . The value of B_q is then kept the same. Otherwise, i.e., if no flit is routed to port S , the value of B_q decreases by 1. At each clock cycle, the flits in the buffer are shifted one slot closer to the head of the queue. A flit reaching the buffer's head is then transmitted through port S .

This approach ensures that flits wait exactly B_q clock cycles before being transmitted through port S , and B_q is always larger than $S_x - 1 - d$ where d is the delay between the arrival of a new flit in port S of router R_q , and the last time a flit was deflected in R_q , thus avoiding the problem discussed above. When B_q is equal to 0, flits are not buffered and are directly transmitted through port S .

Example. Consider a 4×4 IPDeN NoC (see Figure 4.3) and a packet p (red packet) that traverses the network from its origin router $(1;0)$ to its destination $(1;3)$. Let's assume that initially, the value of B_q is 0 in all routers. The packet p is split in three flits sequentially injected to the port S of its origin router (since $x_{q,i}^o = x_{q,i}^d$). Assume that in router $(1;1)$, the flit 1 of p requests the port S at the same time as the flit of a green packet (see Figure 4.3a). In this contention scenario, according to row 2 in Table 4.1, the first red flit is deflected to the port E , and the green flit is routed to the port S (see Figure 4.3a). Because there is a deflection, the variable B_q of router $(1;1)$ is set to $S_x - 1 = 3$. Thus, at the next clock cycle, the flit 2 of p , which is routed to the port S as it does not conflict with any other flow, is buffered in the last position of the buffer and the value of B_q remains unchanged (see Figures 4.3a and 4.3b). One clock cycle later, flit 3 of p requests also the port S of router $(1;1)$. However, if a flit of another packet (blue packet) also requests the port S at the same time as in Figure 4.3c. The flit 3 of p is deflected to the port E and the blue flit is buffered in the last position of the buffer (Figures 4.3b and 4.3c). The value of B_q is thus still equal to 3 since a flit has been buffered at each clock cycle since the last deflection. If no flit requests the port S at the next clock cycle as in Figure 4.3c, the value of B_q becomes 2. Therefore, the next flit to be routed to the port S of router $(1;1)$ (pink flit), is buffered in position number 2 of the buffer (Figures 4.3d and 4.3e). Note that, thanks to the fact that flit 2 of red packet p was buffered in router $(1;1)$, the flits 1, 2, and 3 of p reach the destination router $(1;3)$ in an orderly fashion (see from Figure 4.3d to Figure 4.3f).

4.3 Discussion

4.3.1 Buffer Size Analysis

One of the biggest advantages of IPDeN is that the buffers are small and their sizes do not depend on the set of flows that traverse the network. There is a clear difference with most NoCs that implement VCs or another type of buffering mechanism meant to avoid back-pressure or losing flits during the transmission. For instance, in HopliteBuf, buffers are also added to routers to enforce in-order flit delivery, but those buffers have sizes that depend on the bandwidth requested by the communication flows transmitted through the NoC as well as their source and destination.

In practice, buffers tend to be oversized to ensure they can cope with unforeseen circumstances. In IPDeN, no such resource waste is needed. The size of the buffers depends solely on the network topology, i.e., it is equal to $S_x - 1$, ensuring that the NoC's power and silicon footprint remain small and independent of the specific applications running on the network.

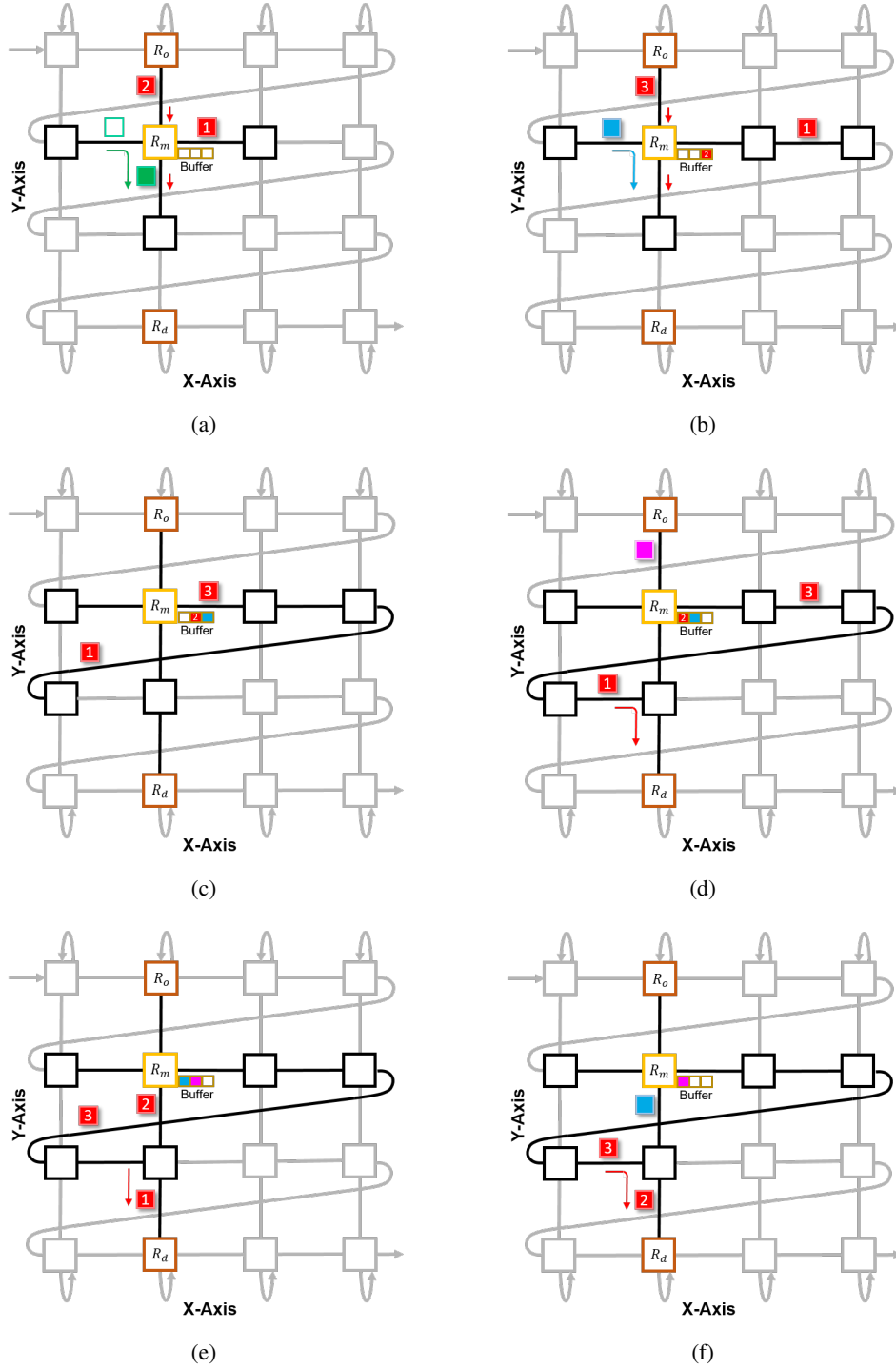


Figure 4.3: Example of how buffers are used in IPDeN.

4.3.2 Livelock and Deadlock Situations

IPDeN is a livelock-free solution. As in HopliteRT*, the circulant topology ensures that flits always progress toward their destination even when they are deflected. Once flits reach their destination routers, they always exit the network via ports PEo_1 and PEo_2 . IPDeN's routing policy also guarantees deadlock-free transmissions. In IPDeN, any flit entering a router and requesting a certain output port may be: (1) routed to the desired port, (2) deflected to an alternative port, (3) or buffered. Under scenarios 1 and 2, the flit never stops progressing towards its destination. Conversely, in scenario 3, the flit is forced to wait inside a buffer before exiting the router. Nonetheless, as discussed in section 4.2.2, it is shifted one slot every clock cycle until reaching the head of the buffer no matter what. Once at the buffer's head, the flit is always routed to the port S of the router and is thus allowed to resume its journey to its destination. A flit is therefore never blocked indefinitely.

4.3.3 Priority Levels

IPDeN's routing policy does not make any distinction between different priority levels, i.e., all flits are treated identically. The main reason behind this design choice is that the worst-case transmission time of all flits will always be the same, whatever their priority. Indeed, in each router of IPDeN, the transmission of a flit may be delayed by a deflection to an alternative port or, if it is not deflected, may be buffered inside the router for a number of clock cycles equal to the extra cost induced by a deflection. Note that buffering must happen without any consideration of the flit's priorities in order to ensure in-order flit delivery. Therefore, the worst-case traversal time will remain the same whether it does or does not support priority-aware routing. Although our new solution does not provide different quality-of-service to different classes of traffic anymore, our network architecture does ensure predictable traversal time for all flits. Moreover, the notion of priority introduced in the previous chapter remains compatible with the IPDeN design. This property can be added to the design to ensure improved average traversal time for high-priority flits. If a high-priority flit traveling through the network reaches routers where no deflection took place in the previous $S_x - 1$ cycles, it will not be buffered. In the worst-case scenario, it will be deflected once every two routers on the Y -axis if conflicting with other high-priority flits. Otherwise, it will always win access to the requested resources and be routed to the desired outputs, leading to improved average traversal time compared to low-priority flits.

4.4 Worst-Case Communication Time Analysis

In this section, we present an analysis for the WCCT between two PEs connected with IPDeN. The WCIT depends on the set of communication flows traversing the network, therefore it can be computed as in Section 3.4.3. The WCTT of a flit of flow $f_{q,i}$ is then computed as

$$wctt_i = h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) \times (1 + cost) + 2 \quad (4.1)$$

where $h_r^i(x_{q,i}^d, y_{q,i}^d)$ is the number of hops made by the flit on the ring until reaching its destination, assuming a zero contention (i.e., that no flit is ever deflected in the network); and $h_b^i(x_{q,i}^d, y_{q,i}^d)$ is the number of hops on the bypasses until its destination, assuming no contention. $cost$ is the maximum number of cycles by which the journey of a flit may be increased due to a single deflection or buffering in one of the routers. The additional two hops in Equation (4.1) account for the injection (at the origin router) and exit (at the destination router) of the flit into and from the network, respectively. We now explain how Equation (4.1) was derived.

It was proved in Section 3.4.1 that $h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) + 2$ is the maximum number of hops a flit must make between its origin and destination if it suffers no deflection and is never buffered in any router along its route (see Equation 3.3). We must thus still prove (using Property 2 and Lemma 21 below) that the maximum total delay that may be suffered by any flit of $f_{q,i}$ due to deflections or buffering is upper bounded by $h_b^i(x_{q,i}^d, y_{q,i}^d) \times cost$. We then prove in Lemma 22 that $cost = S_x - 1$.

Property 2. A flit of flow $f_{q,i}$ may only be deflected in routers in which it requests the port S .

Proof. As discussed in previous sections, a flit of flow $f_{q,i}$ can suffer a deflection if and only if it requests the same output port as other coming flit at the same time. According to IPDeN's routing policy (see Table 4.1), a flit of flow $f_{p,i}$ entering a router R_m by the port W may request the port E or S , while a flit of flow $f_{q,i}$ entering R_m by the port N can only request the port S . Then, flits coming from the port W and requesting the port E travel freely along the X-axis as they do not conflict over the port E with other flits traversing the network. However, if the flit of flow $f_{p,i}$ requests the port S at the same time as the flit of flow $f_{q,i}$, one of the flit wins access to port S and another one is deflected to the port E . Hence, a flit may only be deflected in a router in which it requests the port S and thus, the property is proved. $\square$

Let $n_{def}^i(x_{q,i}^d, y_{q,i}^d)$ and $n_{buf}^i(x_{q,i}^d, y_{q,i}^d)$ be the maximum number of times a flit of $f_{q,i}$ suffers deflections and the maximum number of times it enters a buffer, respectively. The following lemma holds

Lemma 21.

$$n_{def}^i(x_{q,i}^d, y_{q,i}^d) + n_{buf}^i(x_{q,i}^d, y_{q,i}^d) = h_b^i(x_{q,i}^d, y_{q,i}^d), \quad (4.2)$$

Proof. According to Property 2, a flit of flow $f_{q,i}$ may be deflected or buffered as many times as it requests the port S of a router on its route, that is, as many time as the Y-distance between the origin and destination router, i.e., $h_b^i(x_{q,i}^d, y_{q,i}^d)$. Since a flit is either deflected or buffered (but not both) in each such router, we have $n_{def}^i(x_{q,i}^d, y_{q,i}^d) + n_{buf}^i(x_{q,i}^d, y_{q,i}^d) = h_b^i(x_{q,i}^d, y_{q,i}^d)$. $\square$

The additional cost (in terms of clock cycles) added by each deflection or buffering is defined in Lemma 22.

Lemma 22. The maximum cost of a deflection or buffering is upper bounded by $cost = S_x - 1$.

Proof. First, when a flit is deflected in a router R_m , it must do S_x hops instead of 1, before reaching the same router as it would have if it could have used the S port instead. Therefore, each deflection involves an additional cost of $S_x - 1$ hops. Second, the size of the buffer in each router is $S_x - 1$. Hence, the cost of a buffering is *at most* $S_x - 1$. Then, the maximum cost of a deflection or buffering is $S_x - 1$. $\square$

Then, we prove Equation 4.1 in Theorem 2.

Theorem 2.

$$wctt_i = h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) \times S_x + 2$$

Proof. By combining Lemmas 21 and 22, we get that the total additional delay suffered by the flit due to both deflections and buffering is upper bounded by $h_b^i(x_{q,i}^d, y_{q,i}^d) \times (S_x - 1)$. Then, the $wctt_i$ of the flit is determined by the sum of the output of Equation 3.3 and the total additional delay due to deflections and buffering, i.e., $wctt_i = h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) + 2 + h_b^i(x_{q,i}^d, y_{q,i}^d) \times (S_x - 1)$. Hence proving the theorem. $\square$

4.5 Network Latency Discussion

The timing behavior of IPDeN is deterministic. The traversal time analysis proposed in Section 4.4 only utilizes information of the flit under analysis and does not use any information from other flows that may traverse the network. Hence, this analysis is useful for applications with dynamically varying workload and timing properties. IPDeN provides predictable performances to those applications without any need to over-provision or dynamically reconfigure the network. Injection time however may vary with the workload. This could be addressed by adding leaky buckets at each network input to limit the maximum bandwidth that may be claimed by each PE and/or communication flow. However, in Chapter 6, we present a new solution for regulating traffic injection in NoCs.

4.6 Experimental Results

4.6.1 Implementation of IPDeN

We implemented IPDeN with the hardware description language Verilog. We synthesized a single router of IPDeN for flits of 64 bits. The target platform was a Xilinx Kirtex-7 FPGA. A 64-bits IPDeN router required 471 LUTs and 715 Flip-Flops (FFs). This corresponds to between 0.1% and 0.7% of the total number of LUTs available in the target FPGA, respectively.

We compared the hardware resource utilization with HopliteBuf [28], our closest competitor (as it is, as far as we know, the most recent solution for in-order packet delivery inspired by a deflection-based routing policy), and three VC-based NoCs: CONNECT, IDAMC, and ProNoC (see Table 4.2). As already mentioned in Section 3.5.1, according to [67] and [88], a single router of CONNECT and IDAMC requires approximately 1500 and 1300 LUTs, respectively. A ProNoC

router with two VCs requires 1574 LUTs. Comparatively, a router of HopliteBuf requires 402 LUTs. Therefore, IPDeN needs three times less resources than VC-based solutions. However, HopliteBuf is 15% cheaper (in terms of LUTs utilization) than IPDeN. This can mainly be explained by the simpler routing policy of HopliteBuf. However, the buffer size in HopliteBuf depends on the specific applications running on the network. Therefore, if more communication flows must be transmitted through the network, or even if their sources and destinations change, larger buffers are likely to be needed to avoid buffer overflows and thus lose flits during their transmissions. These large buffers not only occupy silicon space but also increase power consumption non-negligibly. Instead, the size of the buffers in IPDeN does not depend on the set of flows traversing the network, but only on the network topology.

Table 4.2: Resources utilization of one router.

NoC	VC-based	HopliteBuf	IPDeN
LUTs	1300-1574	402	471

4.6.2 RTL Simulations

In this section, we provide experimental results by performing cycle-accurate simulations. We used an HDL Verilog implementation of a 4x4 IPDeN and a 4x4 HopliteBuf NoC. We generated sets of flows according to a random traffic pattern. We configured each PE to inject one, two, or three flows into the network. The destination coordinates of each flow were randomly generated using a uniform probability distribution. The inter-arrival time $T_{q,i}$ of each flow was randomly selected within the set $\{100, 200, \dots, 900\}$. The utilization of each PE was randomly chosen within the interval $[U_{bound} - 5\%, U_{bound}]$, where U_{bound} is a parameter of the experiment. The utilization $U_{q,i}$ of each flow was generated using the method described in [8]. The number of flits $C_{q,i}$ in each packet is determined by multiplying the flow's utilization $U_{q,i}$ by $T_{q,i}$. The flit size was set to 64 bits. Each value reported in Figure 4.4a and 4.4b is the result of running 20 experiments.

In Figure 4.4a, we show the average measured communication time for an increasing number of communication flows per PE (i.e., one, two, and three flows per PE) when $U_{bound} = 20\%$. As earlier discussed, the average measured communication time is given by the sum of the average measured traversal time and average injection time. We observe that IPDeN performs better than HopliteBuf in terms of traversal time. That can be explained by the fact that in HopliteBuf, flits can wait long periods of time in the buffer inside each router. Indeed, the number of cycles that flits must wait inside buffers depends on the set of flows that travel through the network. That is, flits may be blocked in the buffers as many cycles as other flows use the port S . In IPDeN, however, the worst-case buffering time in each router is limited to $S_x - 1$ clock cycles, i.e., it is independent of the traffic and only depends on the network topology. As another consequence of the aforementioned property, the average traversal time remains practically the same when the number of flows that are traversing IPDeN increases. On the other hand, HopliteBuf performs better than IPDeN in terms of average injection time. We suspect that can be explained by the fact that HopliteBuf replaces the need for deflection by using bigger buffers in each router instead.

Therefore, less flits use the port E of the routers, and those can be used by PEs to inject new flits in the network instead. Note that, the average injection time decreases when the number of flows injected by PEs increases. It can be explained by the fact that, if the number of flows increases for the same value of U_{bound} , the average utilization of each flow decreases, thereby meaning that the number of flits in the packets in each flow decreases. Therefore, the injection time of all flits of a packet decreases too. Finally, we see that the average measure communication time of packets is more or less 20% smaller than with HopliteBuf.

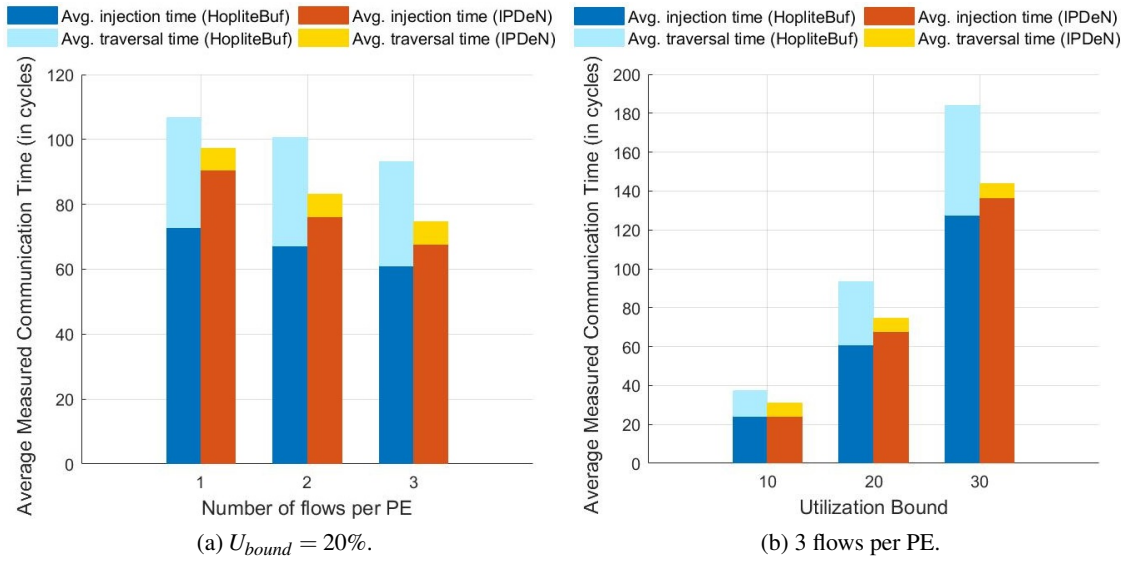


Figure 4.4: Average measured communication time for random traffic.

In Figure 4.4b, we show the average measured communication time for an increasing utilization bound by measuring the average traversal time and injection time. We varied the utilization bound U_{bound} from 10 to 30% by steps of 10. We configured each PE to inject three communication flows into the network. Note that, greater PEs utilization means that each packet contains more flits. Therefore, it is expected that the injection time of a packet increases too when the number of flits it must inject increases (as seen in Figure 4.4b). However, as in the previous experiment, and for the same reasons discussed there, we observe that the average traversal time stays the same in IPDeN when U_{bound} increases. This is not the case for HopliteBuf which sees its traversal time increasing when U_{bound} increases. Therefore, overall, IPDeN has a smaller average communication time and IPDeN performs better than HopliteBuf when the volume of data transmitted increases.

4.6.3 Test Cases

In this section, we use a synthetic test case [86] and a real case study (Orion Crew Exploration Vehicle) [71] to evaluate our proposed NoC architecture. Both test cases are typically used to evaluate Ethernet-based systems, but we adapted them to meet our experimental requirements. Specifically, we assumed that the entire system is consolidated on a single chip and that the computation nodes rely on a NoC to communicate instead of using an Ethernet network. Thus, we

compared the performances of IPDeN with other types of NoCs when these networks must support realistic communication loads.

The configuration details of the test cases are presented in Table 4.3. The origin and destination PEs of each communication flow were randomly generated in both test cases. For the synthetic test case, the traffic flows' parameters were taken from [96]. For the realistic test case, upper and lower bounds on packet size and inter-arrival time were set as in [85]. Then, the packet size and the inter-arrival time associated to each communication flow were randomly selected within the intervals $[S_{min}, S_{max}]$ and $[T_{min}, T_{max}]$, respectively. For both test cases, the flit size was set to 64 bits.

Table 4.3: Configuration details for the test cases.

Test case	Network size	Number of PEs	Number of PEs generating communication flows	Number of flows	S_{min} (Bytes)	S_{max} (Bytes)	T_{min} (Clock cycles)	T_{max} (Clock cycles)
Synthetic	4x4	16	12	46	20	1443	2	25
Orion	6x6	36	31	187	20	1460	4	375

We evaluate IPDeN against HopliteBuf and a generic VCs-based NoC. For HopliteBuf, we consider 128-deep buffers in the routers. For the VCs-based NoC, we consider one-slot VCs. We measured the traversal and communication times of flits of the flow sets defined by the test cases when traversing IPDeN, HopliteBuf and the VCs-based NoC. We identified for each flow the worst measured traversal time (WMTT) and the worst measured communication traversal (WMCT). Moreover, we computed the average measured traversal time (AMTT) and the average measured communication time (AMCT).

Tables 4.4-4.7 show the number of flows, for which IPDeN performs better, the same, or worse than HopliteBuf or the VCs-based NoC. The results are presented for both test cases.

In Tables 4.4 and 4.5, we observe that the WMTT and AMTT of between 59% and 66% of flows are better with IPDeN as compared to HopliteBuf. HopliteBuf performs better than IPDeN for $\approx 20\%$ of flows, and between 14% and 20% of flows have similar traversal times with both NoCs. In terms of communication time (which is the sum of traversal time and injection time), between 50% and 74% of flows have better results with HopliteBuf, while IPDeN performs better for between 26% and 50% of flows. Note that, 73220 flits out of 225995 flits never reach their destination when HopliteBuf was used for the synthetic test case, since the buffers in the routers overflowed even when each router contains a 128-deep buffer. For the realistic test case, the number of lost flits was 79583 out of 301746 flits. This shows that HopliteBuf must be oversized to support realistic communication loads. This is not the case for IPDeN, which only needs a single 3-deep or 5-deep buffer in each router (depending on the test case).

In Tables 4.6 and 4.7, we evaluate IPDeN against the VCs-based NoC. The WMTT, WMCT, and AMCT of more than 50% of the flows are better or the same with IPDeN in contrast to the VCs-based NoC. In term of AMTT, the VCs-based NoC performs better than IPDeN. That is, between 52% and 67% of flows have better results with the VCs-based NoC. This is however

at the cost of much complex logic since VC-based routers require three times more hardware resources than a IPDeN router (see Section 4.6.1).

Tables 4.8 and 4.9 show the number of flows for which IPDeN performs better, the same, or worse than both HopliteBuf and the VCs-based NoC for the synthetic test case and the real-life case study, respectively.

Table 4.4: IPDeN vs HopliteBuf (Synthetic test case).

	IPDeN<HopliteBuf	IPDeN=HopliteBuf	IPDeN>HopliteBuf
WMTT	27 out of 46 (58.7%)	9 out of 46 (19.6%)	10 out of 46 (21.7%)
WMCT	19 out of 46 (41.3%)	0 out of 46 (0%)	27 out of 46 (58.7%)
AMTT	28 out of 46 (60.9%)	8 out of 46 (17.4%)	10 out of 46 (21.7%)
AMCT	23 out of 46 (50%)	0 out of 46 (0%)	23 out of 46 (50%)

Table 4.5: IPDeN vs HopliteBuf (Orion test case).

	IPDeN<HopliteBuf	IPDeN=HopliteBuf	IPDeN>HopliteBuf
WMTT	124 out of 187 (66.3%)	27 out of 187 (14.4%)	36 out of 187 (19.3%)
WMCT	52 out of 187 (27.8%)	0 out of 187 (0%)	135 out of 187 (72.2%)
AMTT	120 out of 187 (64.2%)	27 out of 187 (14.4%)	40 out of 187 (21.4%)
AMCT	49 out of 187 (26.2%)	0 out of 187 (0%)	138 out of 187 (73.8%)

Table 4.6: IPDeN vs VCs-based NoC (Synthetic test case).

	IPDeN<VCs-NoC	IPDeN=VCs-NoC	IPDeN>VCs-NoC
WMTT	19 out of 46 (41.3 %)	8 out of 46 (17.4%)	19 out of 46 (41.3%)
WMCT	26 out of 46 (56.5%)	0 out of 46 (0%)	20 out of 46 (43.5%)
AMTT	14 out of 46 (30.4%)	8 out of 46 (17.4%)	24 out of 46 (52.2%)
AMCT	24 out of 46 (52.2%)	0 out of 46 (0%)	22 out of 46 (47.8%)

Table 4.7: IPDeN vs VCs-based NoC (Orion test case).

	IPDeN < VCs-NoC	IPDeN = VCs-NoC	IPDeN > VCs-NoC
WMTT	80 out of 187 (42.78%)	14 out of 187 (7.48%)	93 out of 187 (49.73%)
WMCT	104 out of 187 (55.6%)	0 out of 187 (0%)	83 out of 187 (44.4%)
AMTT	47 out of 187 (25.1%)	14 out of 187 (7.5%)	126 out of 187 (67.4%)
AMCT	96 out of 187 (51.3%)	0 out of 187 (0%)	91 out of 187 (48.7%)

Table 4.8: IPDeN vs HopliteBuf and IPDeN vs VCs-based NoC (Synthetic test case).

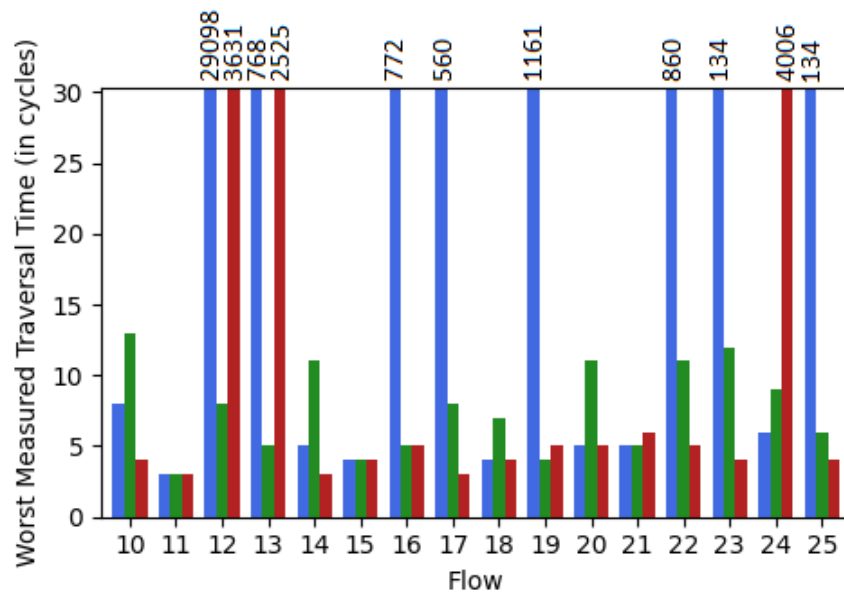
	IPDeN < HopliteBuf and IPDeN < VCs-NoC	IPDeN = HopliteBuf = VCs-NoC	IPDeN > HopliteBuf and IPDeN > VCs-NoC
WMTT	13 out of 46 (28.3%)	4 out of 46 (8.7%)	8 out of 46 (17.4%)
WMCT	13 out of 46 (28.3%)	0 out of 46 (0%)	14 out of 46 (30.4%)
AMTT	11 out of 46 (23.9%)	4 out of 46 (8.7%)	10 out of 46 (21.7%)
AMCT	14 out of 46 (30.4%)	0 out of 46 (0%)	13 out of 46 (28.3%)

Table 4.9: IPDeN vs HopliteBuf and IPDeN vs VCs-based NoC (Orion test case).

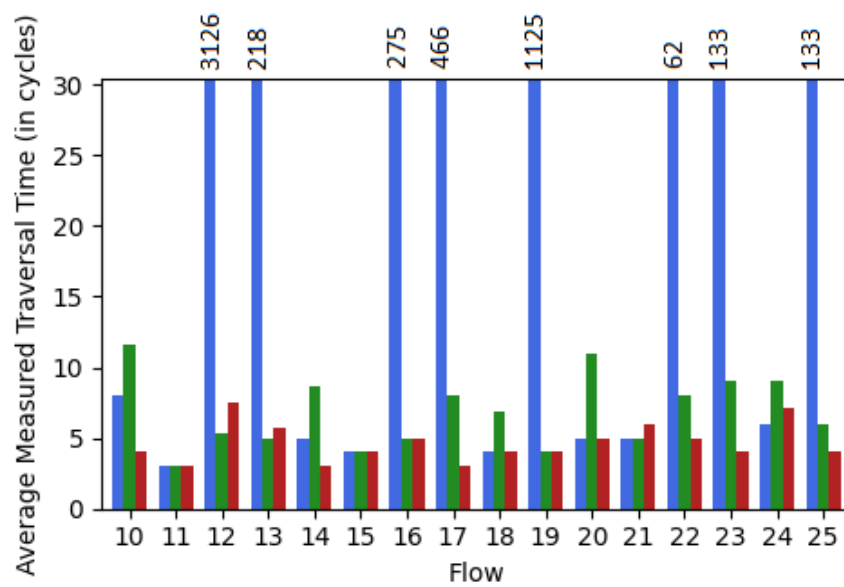
	IPDeN < HopliteBuf and IPDeN < VCs-NoC	IPDeN = HopliteBuf = VCs-NoC	IPDeN > HopliteBuf and IPDeN > VCs-NoC
WMTT	62 out of 187 (33.2%)	11 out of 187 (5.9%)	28 out of 187 (15%)
WMCT	40 out of 187 (21.4%)	0 out of 187 (0%)	71 out of 187 (38%)
AMTT	34 out of 187 (18.2%)	11 out of 187 (5.9%)	37 out of 187 (19.8%)
AMCT	37 out of 187 (19.8%)	0 out of 187 (0%)	79 out of 187 (42.2%)

Figures 4.5 and 4.6 present plots showing the WMTT, AMTT, WMCT, and AMCT of some of the flows for both test cases. The segments were selected to show when IPDeN performs better, the same or worse than HopliteBuf and the VCs-based NoC. The results for all communication flows can be found in Appendix B.

■ Avg. traversal time (HBuf)
 ■ Avg. traversal time (IPDeN)
 ■ Avg. traversal time (VCs-based NoC)

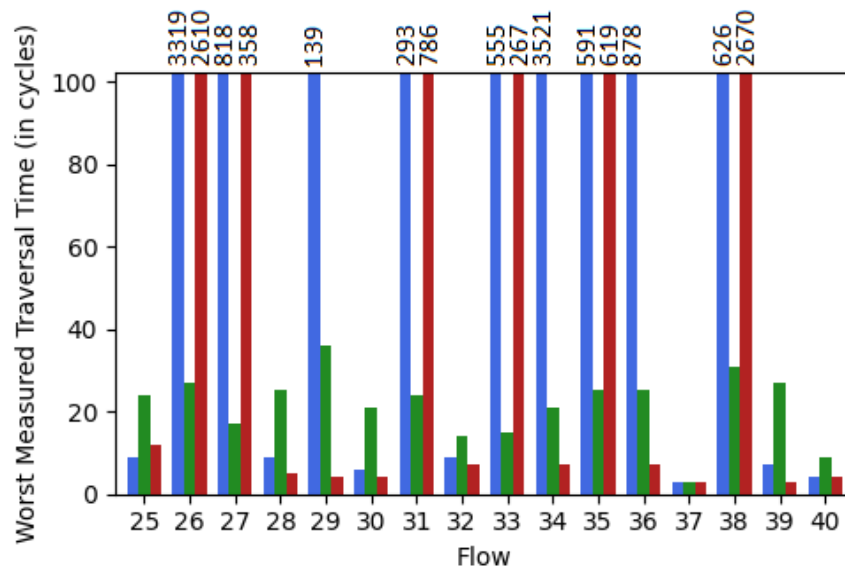


(a) WMTT (Synthetic test case).

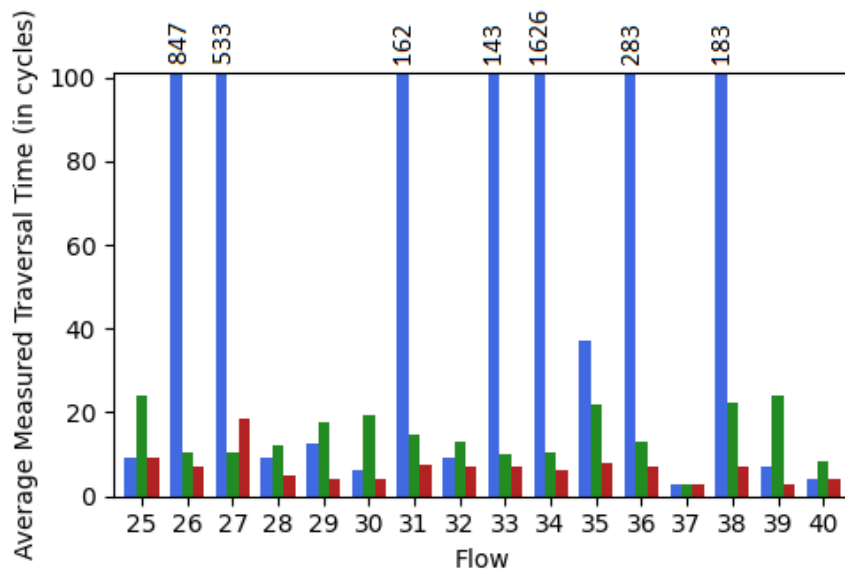


(b) AMTT (Synthetic test case).

Figure 4.5: Experimental results for Synthetic and Orion test case (traversal time).

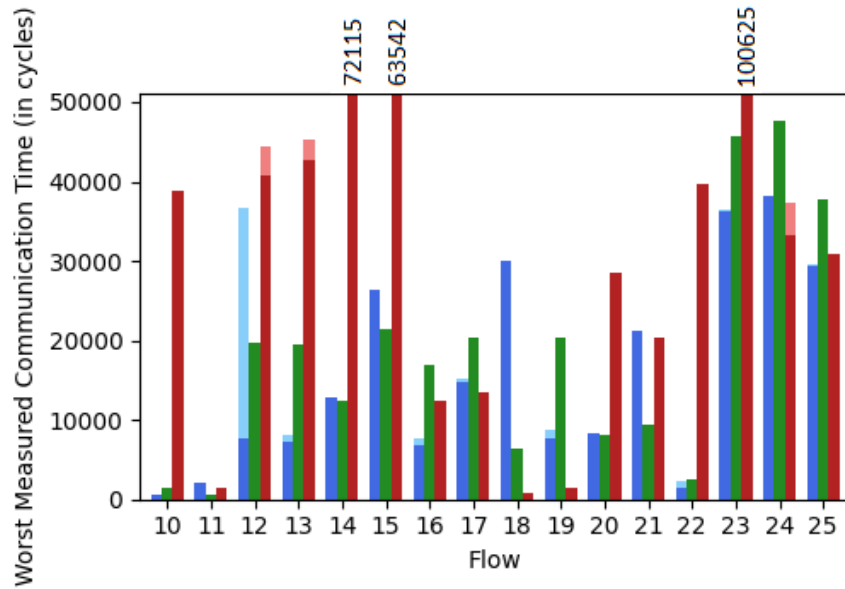
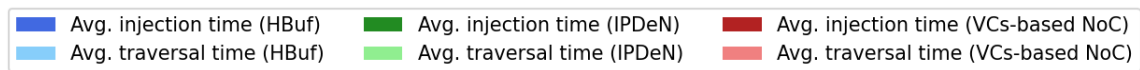


(c) WMTT (Orion test case).

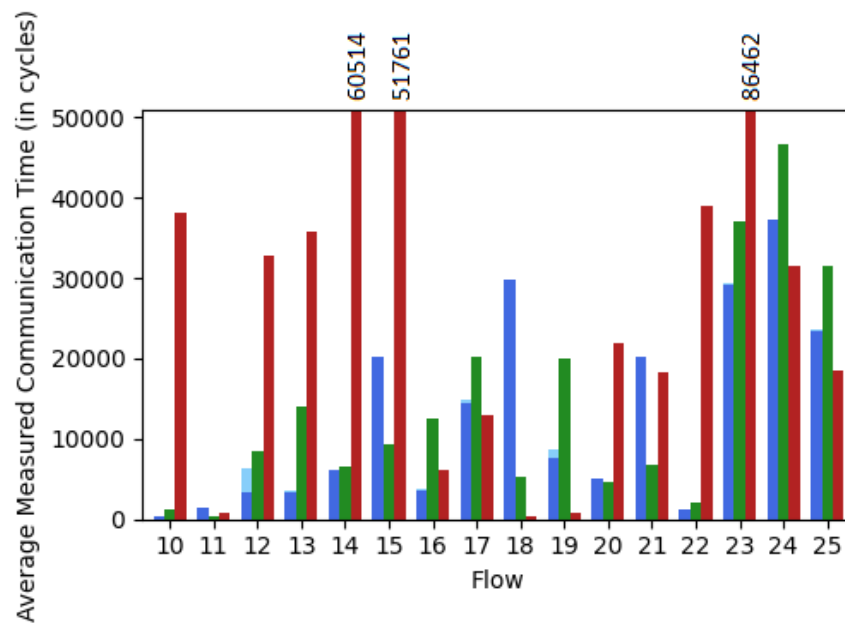


(d) AMTT (Orion test case).

Figure 4.5: Experimental results for Synthetic and Orion test case (traversal time) (cont.).

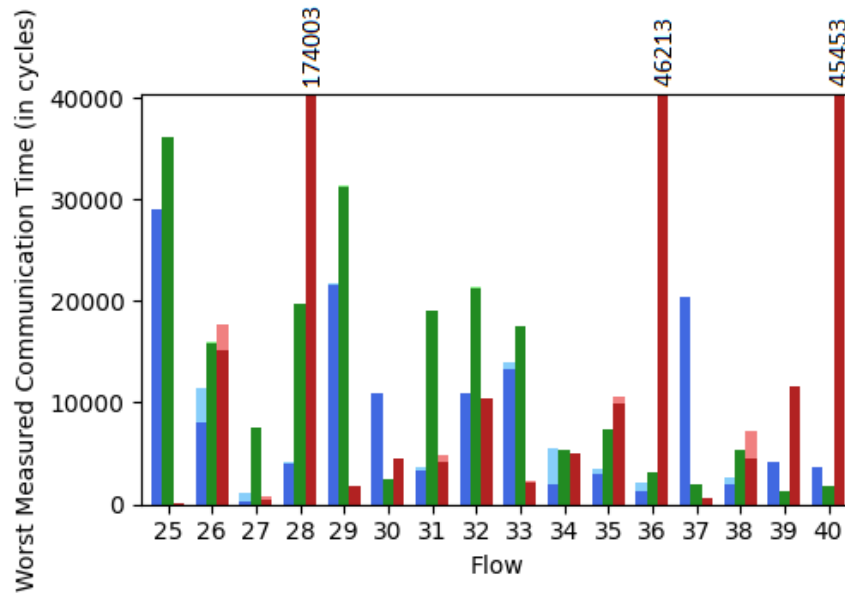


(a) WCCT (Synthetic test case).

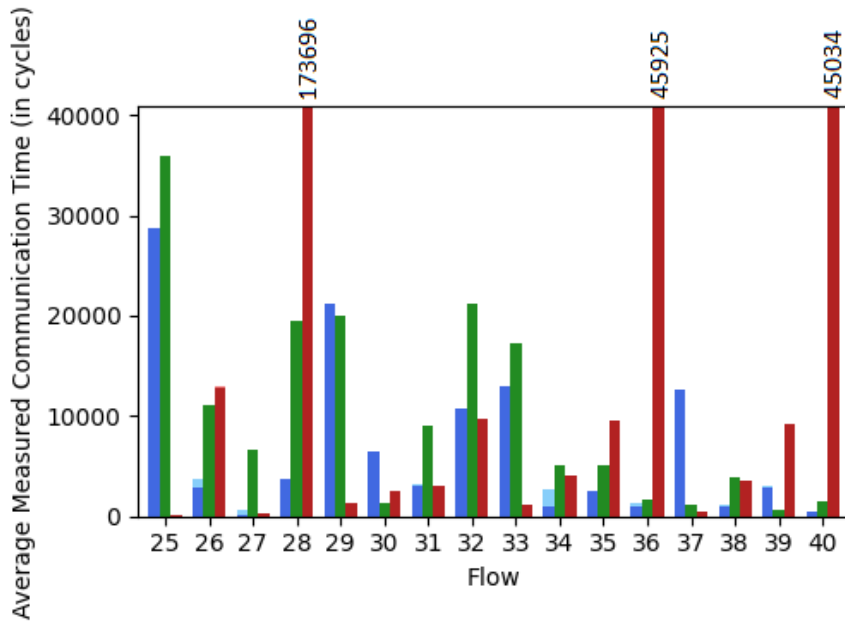


(b) AMCT (Synthetic test case).

Figure 4.6: Experimental results for Synthetic and Orion test case (communication time).



(c) WMCT (Orion test case).



(d) AMCT (Orion test case).

Figure 4.6: Experimental results for Synthetic and Orion test case (communication time) (cont.).

4.7 Chapter Summary

In this chapter, we introduced IPDeN, an evolution of HopliteRT* NoC that ensures flits reach their destinations in the same order they were injected into the network by implementing a new adaptive buffering mechanism in each router. Unlike SotA solutions, we demonstrated that the number and size of the buffers in IPDeN depend solely on the network topology, ensuring a small silicon and power footprint independent of specific applications running on the network. The solution allows for predictable performance for applications with dynamically varying workloads and timing properties, without any need to over-provision or dynamically reconfigure the network. Furthermore, we showed that the WCTT of flits across the network is deterministic and solely depends on the properties of the communication flow under analysis. Through cycle-accurate simulations on a Xilinx FPGA, we compared IPDeN with HopliteBuf, which is, as far as the authors know, the most recent solution for in-order packet delivery inspired by a deflection-based routing policy. The results revealed that IPDeN outperforms HopliteBuf in terms of network communication latency.

Chapter 5

Improving Average Worst-Case Traversal Time

IPDeN's routing policy and design ensure deterministic timing behavior. However, one of the downsides of IPDeN comes from its literal use of a deflection-based routing policy, that is, in contention scenarios, flits are always forced to deflect to alternative routes while traversing the network. Moreover, to ensure in-order flit delivery at the destination, flits that are not deflected are instead frequently buffered within the routers. The following example illustrates such behavior.

Example. Consider a 4x4 IPDeN NoC and a red packet split into 2 flits, sequentially injected at the origin router R_o , with destination router R_d (see Figure 5.1). Now, suppose there is a green flit that conflicts with flit 1 of the red packet over port S of router R_m (see Figure 5.1a). According to IPDeN's routing policy (row 2 Table 4.1), the green flit is routed to port S of R_m , while flit 1 of the red packet is deflected to port E . The deflected flit will then travel along the horizontal dimension (X-axis) until it reaches the same router it would have reached if it had not suffered a deflection (see from Figure 5.1b) to Figure 5.1f). In the next cycle (see Figure 5.1b), flit 2 of the red packet reaches R_m and, to guarantee the in-order-flit delivery property, it is buffered at the last position of the buffer, even if it does not conflict with any other flit over port S of R_m . Note that the buffer delays flit 2 as many clock cycles as needed to ensure that flits arrive at their destination in an orderly fashion.

In IPDeN, flits may experience an increase in their average traversal time, as each deflection and buffering adds an additional cost. However, we believe that flits should not always be deflected when conflicting, as they may have the opportunity to use the requested output port during one of the next clock cycles if they remain in the routers instead of being deflected. The idea of introducing small fixed-size buffers in deflection-based NoCs can be used to reduce the number of deflections that flits can suffer. This would allow to reduce the average cost (in terms of clock cycles) generated by deflections when flits traverse the network. The trick however is to keep the buffering cost lower than what was saved by avoiding deflection, but also to keep the maximum cost of buffering independent of the specific application running in the network.

In this chapter, we propose IPDeN 2.0, an improved version of IPDeN that addresses this

matter. It supports a routing policy based on selective deflection and buffering of flits to optimize the average traversal time through the NoC while maintaining a deterministic upper-bound on the worst-case traversal time, and keeping the in-order flit delivery property of IPDeN.

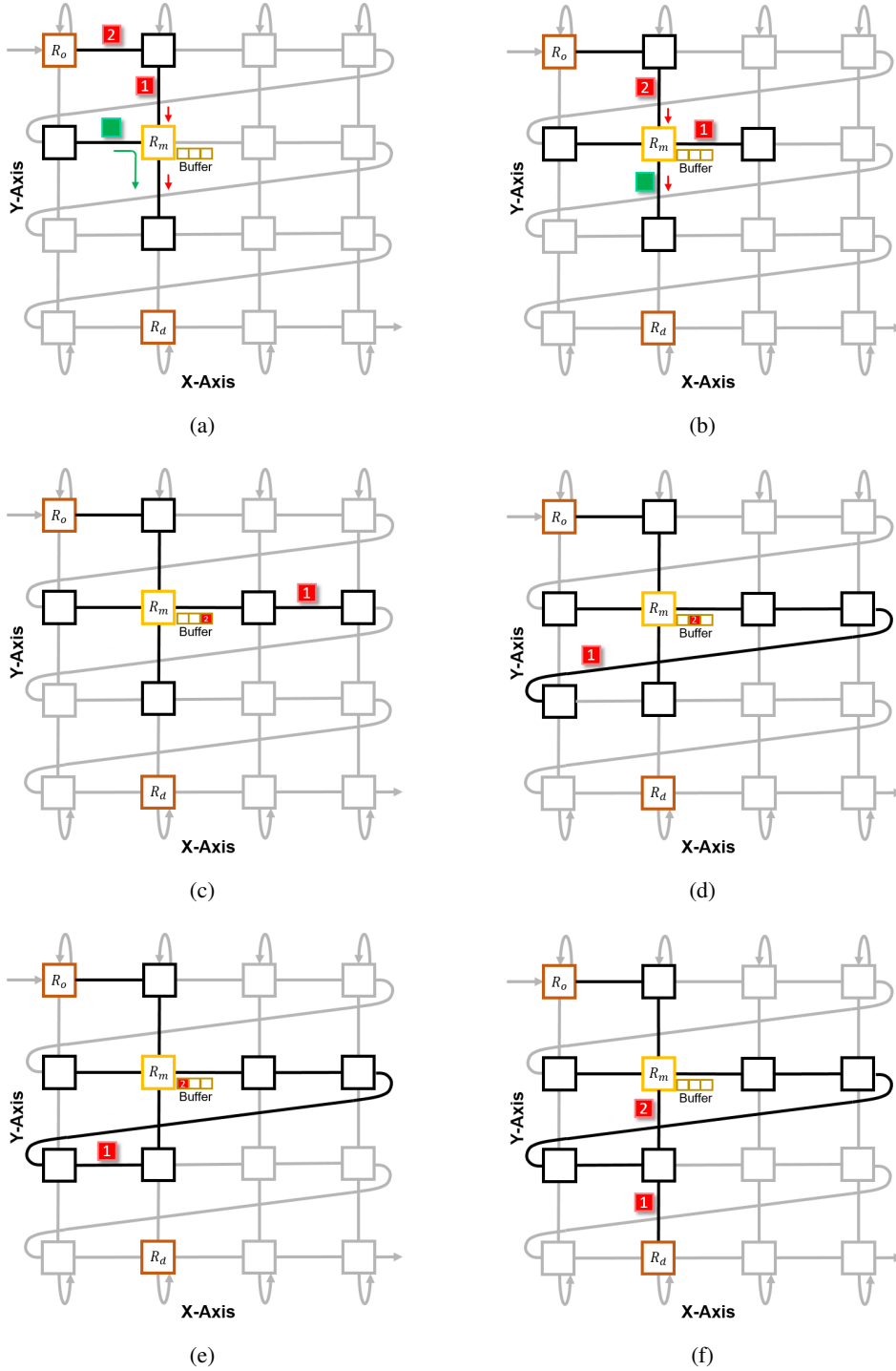


Figure 5.1: Example of the penalties for deflecting and buffering flits in IPDeN.

The organization of the chapter is as follows. Section 5.1 confirms the system model. Section 5.2 proposes a new router architecture associated to a new routing policy, and Section 5.3 analyzes the bound on the WCTT of packets transmitted over IPDeN 2.0. In Section 5.4, we compare IPDeN 2.0 to the SotA in terms of hardware resource requirements and communication time.

The content of this chapter was previously published in the following paper:

- **Y. Ribot González**, G. Nelissen, and E. Tovar. IPDeN 2.0: Real-time NoC with selective flit deflection and buffering. In Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS'23), pp. 87–98, 2023, doi:10.1145/3575757.3575758.

5.1 System Model

We assume a system model identical to that of IPDeN (refer to Section 4.1).

5.2 IPDeN 2.0 Design

5.2.1 Router Architecture

IPDeN 2.0 supports exactly the same network topology as IPDeN, i.e., a ring circulant topology with parameters $(N; g1, g2)$ (see Figure 3.3). The new routing policy of IPDeN 2.0, along with its subsequent analysis, assumes that each PE decides which input port to use for injecting a flit based on the coordinates of the flit's destination in the network. That is, Property 1, presented in Section 4.2.1, holds for IPDeN 2.0.

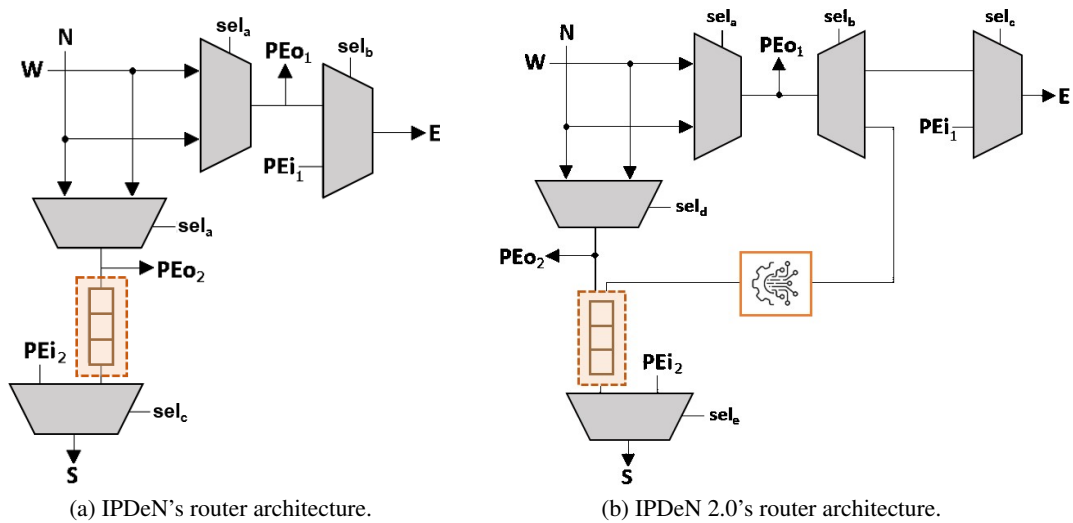


Figure 5.2: Architecture of the routers in IPDeN and IPDeN 2.0 NoCs.

Similar to IPDeN, each router of IPDeN 2.0 contains a small buffer of size $S_x - 1$ (shaded component in Figure 5.2b). The main difference between IPDeN and IPDeN 2.0 is in how decisions about what flits must be buffered are made. Section 5.2.2 discusses this matter in more detail.

5.2.2 Routing Policy

The basics of the routing policy of IPDeN 2.0 is the same as that of IPDeN. Both NoCs deal with contention scenarios using an internal buffer of size $S_x - 1$ in each router (see Figures 5.2a and 5.2b). The slots of the router are numbered from 1 to $S_x - 1$. However, IPDeN 2.0 differs from IPDeN mainly by how that buffer is used and how the router decides which flits should be buffered and which should be deflected to output port E .

To better grasp the difference between IPDeN and IPDeN 2.0, we first remind how IPDeN uses its internal buffers. Each router R_q of IPDeN has a variable B_q pointing to one of the slots of the buffer. After each deflection, B_q is set to $S_x - 1$, pointing to the last slot of the buffer. When a flit is routed to port S on each clock cycle, it is stored at the buffer position pointed to by B_q and the value of B_q remains the same. In contrast, if no flit is routed to port S , the value of B_q decreases by 1. At each clock cycle, the flits in the buffer are shifted one slot closer to the head of the queue. A flit that reaches the head of the buffer is then transmitted through port S . We illustrate that behavior in an example below, which will be revisited later to show the difference between IPDeN and IPDeN 2.0.

Example. Consider a 4×4 IPDeN NoC (see Figure 5.3). Then, each router contains a 3-slot buffer. Suppose that a red flit enters a router R_m via port W and competes for the port S with a green flit coming from the port N (see Figure 5.3a). Assume that the value of B_q is initially 0 for all routers. In this contention scenario, according to IPDeN's routing policy, the red flit wins access to port S of R_m and the green flit is deflected to the port E (see Figure 5.3b). Since there is a deflection, the variable B_q of the router R_m is set to $S_x - 1 = 3$. Later, two new flits (blue and pink flits in Figure 5.3b) reach router R_m and simultaneously request its port S . The pink flit is then deflected to the port E of R_m (see Figure 5.3c). The blue flit is buffered in the last position of the buffer in router R_m and the value of B_q remains unchanged, i.e., $B_q = 3$ (see Figure 5.3c). Next, a yellow and a brown flits conflict over the port S of R_m (see Figure 5.3c). The brown flit is deflected to the port E , the blue flit is moved one slot closer to the head of the queue (i.e., it is now in slot 2 of the buffer), and the yellow flit is buffered in the last position of the buffer (see Figure 5.3d). Thus, the value of B_q is still equal to 3 as a new deflection occurs. Finally, an orange flit coming from port W of R_m requests port S at the same time as a purple flit entering via port N (see Figure 5.3d). In this new contention scenario, the purple flit is deflected to port E and the flits in the buffer are shifted one slot closer to the head of the buffer. That is, the blue and yellow flits are shifted to slots 1 and 2 of the buffer, respectively. The orange flit is then buffered in the last slot of the buffer (see Figure 5.3e). In the next clock cycle, the blue flit is routed through port S .

Note the two following properties of IPDeN. In case of contention, it always deflects one flit, and it nearly always buffers the other flit at the last position of the internal buffer of the router.

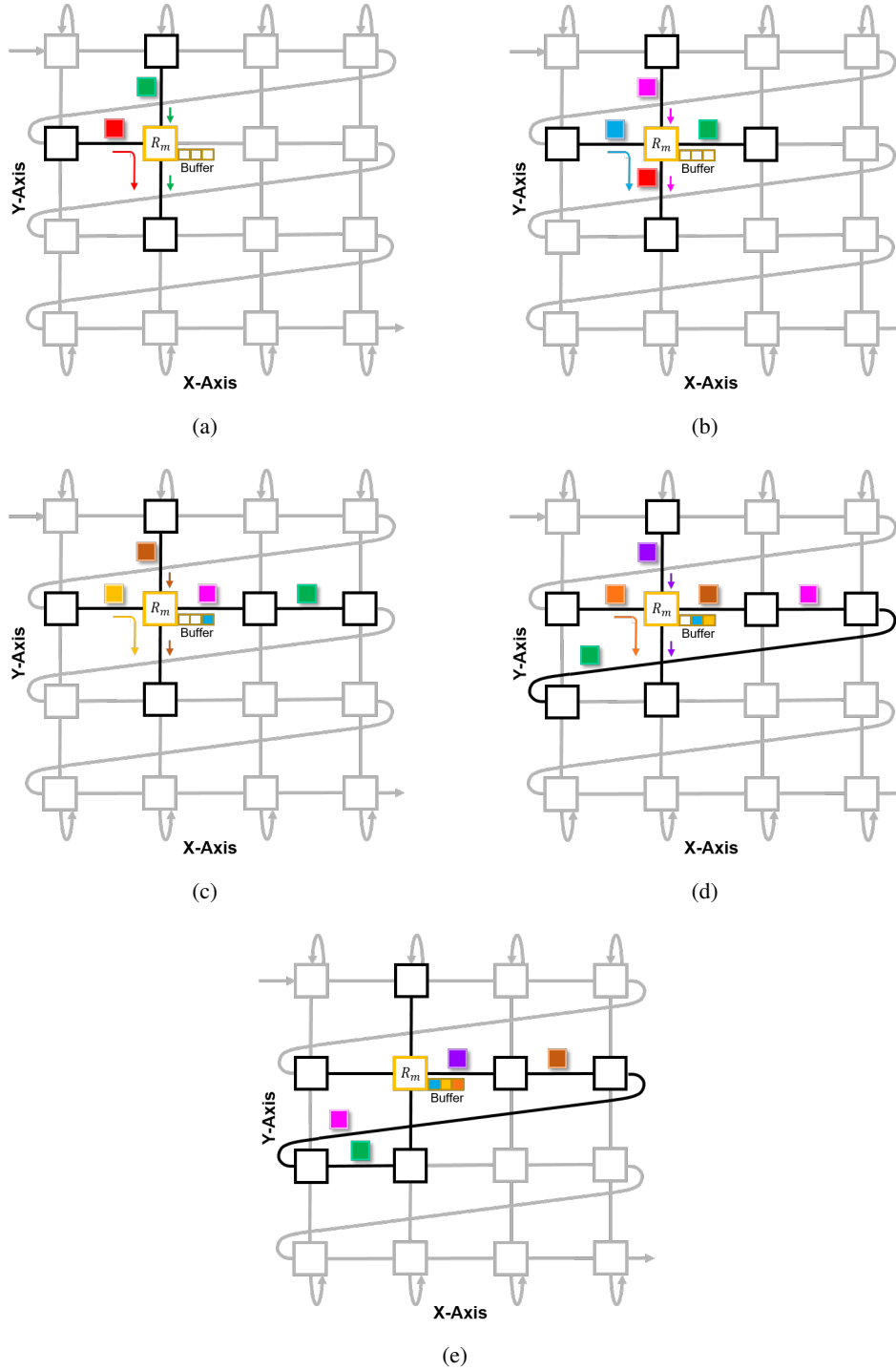


Figure 5.3: Example of IPDeN's routing policy.

IPDeN 2.0, alters the routing policy as follows. In case of contention, no flit is deflected unless the internal buffer is full, and flits buffered in the router are always placed at the first empty spot in the buffer. Therefore, IPDeN 2.0 does not use a variable B_q as in IPDeN. Instead, it used a pointer P_q that points to the first empty slot in the inter buffer of router R_q , e.g. P_q points to slot 1 when

the buffer is empty, and P_q takes the value S_x when the buffer is full.

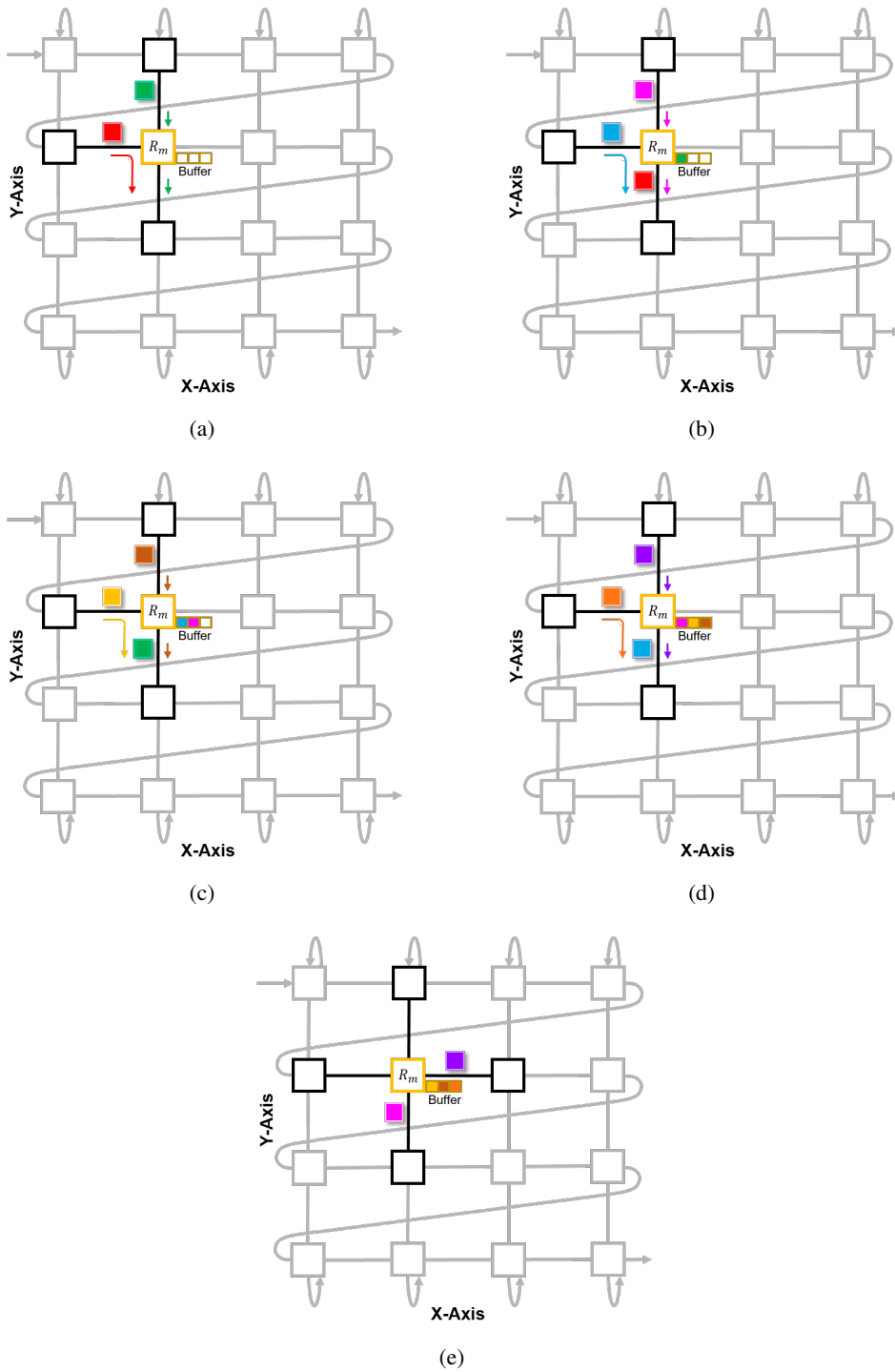


Figure 5.4: Example of IPDeN 2.0's routing policy.

Example. Consider a 4×4 IPDeN 2.0 NoC with exactly the same properties as the network used in the previous example. Also, consider the same scenarios analyzed there. The red flit enters router R_m via port W and competes for port S with a green flit that enters via port N (see

Figure 5.4a). Since the buffer in R_m is empty, P_m points to slot 1. Then, the red flit wins access to port S of R_m and the green flit is buffered in slot 1 of the buffer (see Figure 5.4b). P_m now points to slot 2, which is the first empty slot in the buffer. At the next clock cycle, when the blue and pink flits reach router R_m (coming from ports W and N , respectively) and simultaneously request the port S (see Fig 5.4b), the flit buffered in R_m (i.e., the green flit) is routed to port S (i.e., any flit in slot 1 of a buffer has the highest priority for using the router's port S over any flit reaching the router through one of its input ports). The blue and pink flits are then buffered in slots 1 and 2 of the buffer, respectively (see Figure 5.4c). The slot pointed to by P_m is now 3. Next, the yellow and brown flits (coming from ports W and N , respectively) conflict over the port S of R_m . The blue flit is routed to port S and the pink flit is moved to slot 1 of the buffer. The yellow and brown flits are then buffered in slots 2 and 3 of the buffer, respectively (see Figure 5.4d). Finally, the orange flit (coming from port W) conflicts for the port S of R_m with the purple flit (coming from port N). The pink flit (which was buffered in slot 1 of the buffer) is routed through port S . The flits remaining in the buffer are shifted one slot closer to the head of the buffer. That is, the yellow and brown flits are now in slots 1 and 2 of the buffer, respectively. Since there is only one empty slot in the buffer, the orange flit is buffered at the last slot, and the purple flit is deflected to port E (see Figure 5.4e).

As shown in the above examples, in IPDeN, flits that do not win the right to use port S are always deflected to port E . The additional cost (in terms of clock cycles) added to the traversal time by each deflection is $S_x - 1$ (see Section 4.4). Buffers are then used to delay flits by as many clock cycles as necessary to prevent a flit from overtaking another flit that belongs to the same packet and was previously deflected in the same router, thereby ensuring in-order flit delivery at the destination. In IPDeN 2.0, however, flits are not always deflected in contention scenarios. Buffering is the preferred option. Since the buffer size in IPDeN 2.0 is $S_x - 1$, in the worst case, the additional delay that a flit can suffer when buffered is still $S_x - 1$, the same as in IPDeN. However, in most cases, flits will be buffered in positions closer to slot 1 instead of being buffered in the last slot or being deflected as in IPDeN. Then, flits see their average traversal time reduced in IPDeN 2.0 as compared to IPDeN. The buffers in IPDeN 2.0 are used to avoid the additional cost of deflecting and thus reduce the actual traversal-time of flits. Note that, when a flit is buffered instead of deflected, PE can inject a new flit via PEi_1 on port E since it is then kept free. Therefore, the IPDeN 2.0 approach also has a positive effect on the injection time of flits.

The routing policy of IPDeN 2.0 is summarized in Table 5.1. It consists of 23 cases covering all the possible combinations of output ports requested by flits arriving on inputs ports W , S , PEi_1 , and PEi_2 , and whether the router internal buffer is full, empty or partially filled. The text highlighted in red indicates the variations from IPDeN's routing policy.

In Table 5.1, row 1 shows the case where no contention occurs. When a flit entering a router R_q by the port W conflicts over the port S with a flit entering by the port N , three different situations may then arise. **(1)** The buffer inside R_q is empty (i.e., $P_q = 1$): the flit entering R_q by the port W is routed to the port S , while the flit coming from the port N is buffered in the first position of the buffer (row 2 in Table 5.1). P_q is set to 2. **(2)** The buffer is not empty but not full (i.e., $2 \leq P_q < S_x$): In other words, there is at least one flit buffered in R_q and at most $S_x - 2$ flits. The flit

buffered in slot 1 of the buffer is routed to the port S . If there is more than one flit in the buffer, all the remaining flits are shifted one slot closer to the head of the buffer. Then, the flits entering R_q via ports W and N are buffered in the first and second empty slots of the buffer, respectively. The value of the pointer P_q is then increased by 1 to point to the first empty slot in the buffer (row 3 in Table 5.1). (3) The buffer is full (i.e., $P_q = S_x$): The flit buffered in slot 1 of the buffer is routed to the port S . The flits remaining in the buffer are shifted one slot closer to the head of the buffer. Then, the flit coming from the port W is buffered in the last position of the buffer and the flit coming from the port N is deflected to the port E of the router. The value of P_q remains unchanged (i.e., $P_q = S_x$ as the buffer is still full) (see row 4 in Table 5.1). The deflected flit must then travel along the X -axis until it reaches the same router as it would have if it had not been deflected. Once in that router, it will request the port S again to continue traveling along the Y -axis. From row 5 to row 23 in Table 5.1, we show all cases where the PE competes for the ports S and E .

Table 5.1: Routing Table of IPDeN 2.0.

Case	Input requests	Routing decisions	Explanation
1	$W \rightarrow E + N \rightarrow S$	$W \rightarrow E + N \rightarrow S$	No contention.
2	$W \rightarrow S + N \rightarrow S + \text{buffer is empty } (P_q = 1)$	$W \rightarrow S + N \rightarrow \text{buffer}$	Conflict over the S port. $W \rightarrow S$ requests wins. N flit is buffered.
3	$W \rightarrow S + N \rightarrow S + \text{buffer not empty nor full } (2 \leq P_q < S_x)$	$W \rightarrow \text{buffer} + N \rightarrow \text{buffer}$	Conflict over the S port. N and W flits are buffered.
4	$W \rightarrow S + N \rightarrow S + \text{buffer is full } (P_q = S_x)$	$W \rightarrow \text{buffer} + N \rightarrow E$	Conflict over the S port. W flit is buffered and N flit is deflected.
5	$N \rightarrow S + PE_1 \rightarrow E + \text{buffer is empty } (P_q = 1)$	$N \rightarrow S + PE_1 \rightarrow E$	No contention.
6	$N \rightarrow S + PE_1 \rightarrow E + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$N \rightarrow \text{buffer} + PE_1 \rightarrow E$	N flit is buffered.
7	$N \rightarrow S + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$N \rightarrow S$	PE_2 cannot inject its flit.
8	$N \rightarrow S + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$N \rightarrow \text{buffer}$	N flit is buffered. PE_2 cannot inject its flit.
9	$N \rightarrow S + PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$N \rightarrow S + PE_1 \rightarrow E$	PE_2 cannot inject its flit.
10	$N \rightarrow S + PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$N \rightarrow \text{buffer} + PE_1 \rightarrow E$	N flit is buffered. PE_2 cannot inject its flit.
11	$W \rightarrow S + PE_1 \rightarrow E + \text{buffer is empty } (P_q = 1)$	$W \rightarrow S + PE_1 \rightarrow E$	No contention.
12	$W \rightarrow S + PE_1 \rightarrow E + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$W \rightarrow \text{buffer} + PE_1 \rightarrow E$	W flit is buffered.
13	$W \rightarrow S + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$W \rightarrow S$	PE_2 cannot inject its flit.
14	$W \rightarrow S + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$W \rightarrow \text{buffer}$	W flit is buffered. PE_2 cannot inject its flit.
15	$W \rightarrow S + PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$W \rightarrow S + PE_1 \rightarrow E$	PE_2 cannot inject its flit.
16	$W \rightarrow S + PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$W \rightarrow \text{buffer} + PE_1 \rightarrow E$	W flit is buffered. PE_2 cannot inject its flit.
17	$W \rightarrow E + PE_1 \rightarrow E$	$W \rightarrow E$	PE_1 cannot inject its flit.
18	$W \rightarrow E + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$W \rightarrow E + PE_2 \rightarrow S$	No contention.
19	$W \rightarrow E + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$W \rightarrow E$	PE_2 cannot inject its flit.
20	$W \rightarrow E + PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$W \rightarrow E + PE_2 \rightarrow S$	PE_1 cannot inject its flit.
21	$W \rightarrow E + PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$W \rightarrow E$	PE_1 and PE_2 cannot inject their flits.
22	$PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer is empty } (P_q = 1)$	$PE_1 \rightarrow E + PE_2 \rightarrow S$	No contention.
23	$PE_1 \rightarrow E + PE_2 \rightarrow S + \text{buffer not empty } (2 \leq P_q \leq S_x)$	$PE_1 \rightarrow E$	PE_2 cannot inject its flit.

5.2.3 In-order flit delivery

In IPDeN 2.0, as discussed above, a flit reaching a router R_q and requesting to output via the S port to reach a router R_{q+1} may suffer a deflection if the buffer within R_q is full. Then, the flit must travel along the X-axis until it reaches the same router it would have reached if it had not been deflected (i.e., router R_{q+1}). The cost of a deflection in router R_q is exactly the same as the maximum cost of buffering in R_q . Thus, the deflected flit reaches the next router R_{q+1} at the same time as the flit that contended with it over port S of R_q and was buffered in slot $S_x - 1$ of the buffer, i.e., $S_x - 1$ clock cycles after deflecting.

Accordingly, flits reach R_{q+1} in the same order as they entered R_q . That is, flits reaching R_q in clock cycle clk_q reach router R_{q+1} before any flit that entered R_q in a clock cycle $clk_q + i, \forall i \in \mathbb{N} : i \geq 1$. IPDeN 2.0 thus guarantees in-order flit delivery at the destination, i.e., it ensures that flits from the same packet that were injected into the network one after the other reach their destination in the same order in which they were injected. Therefore, the buffers in IPDeN 2.0 still ensure the in-order flits delivery property introduced in IPDeN.

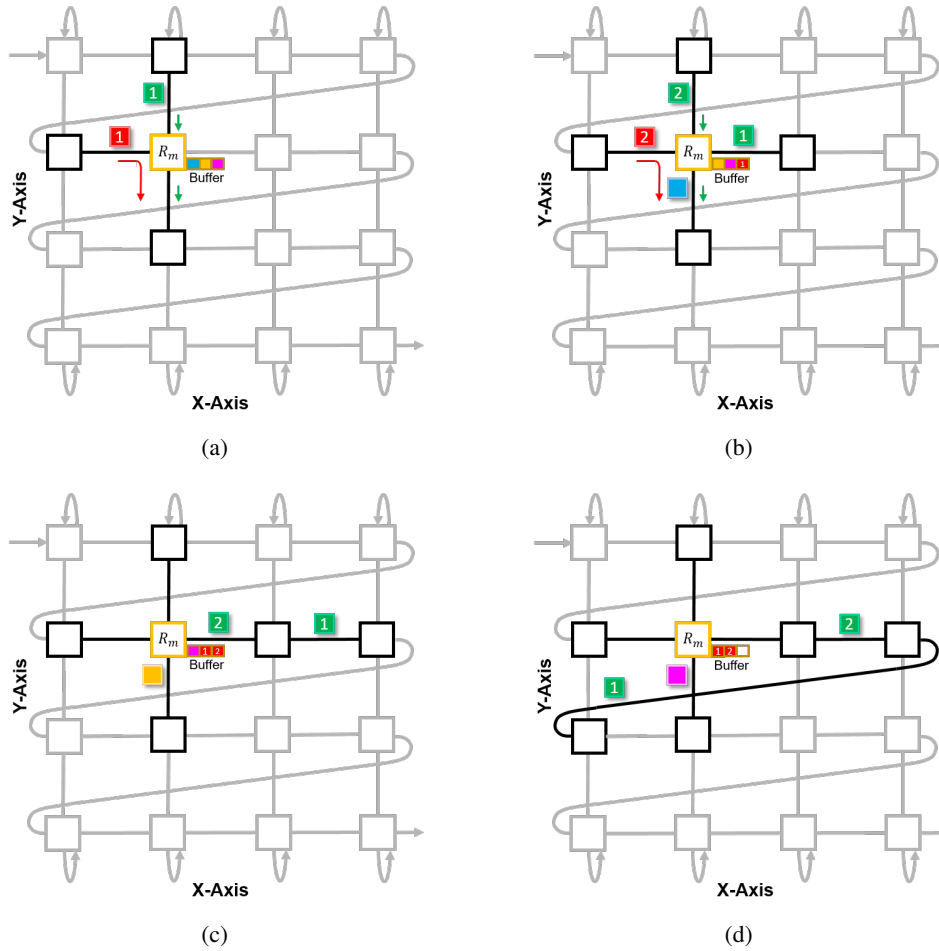


Figure 5.5: Example of in-order flits delivery property in IPDeN 2.0.

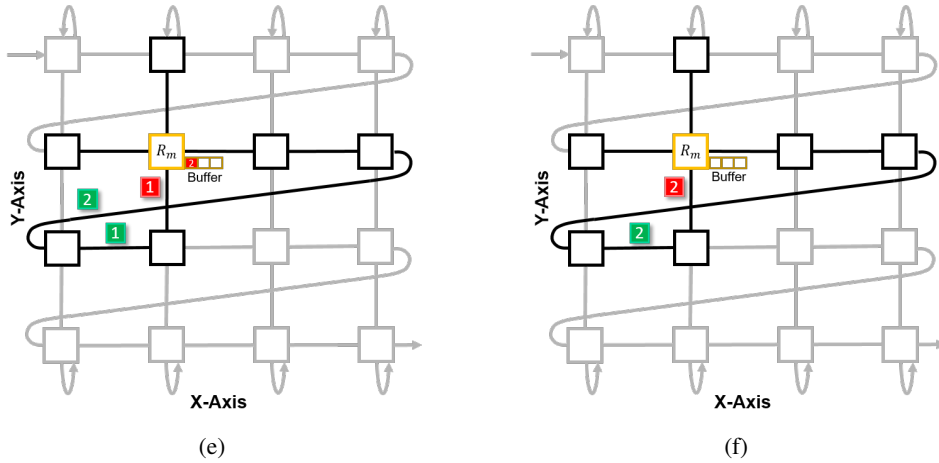


Figure 5.5: Example of in-order flits delivery property in IPDeN 2.0 (cont.).

Example. Consider a 4×4 IPDeN 2.0 NoC (see Figure 5.5). Consider also a packet p (red packet) of flow $f_{l,i}$ traversing the network from its origin router R_o at coordinates $(0;1)$ to its destination router R_d at coordinates $(1;2)$. The packet p is split into two flits that are injected sequentially at port E of R_o (since $x_{l,i}^o \neq x_{l,i}^d$). In the router at coordinates $(1;1)$, flit 1 of p requests the port S at the same time as a flit of a green packet (see Figure 5.5a). Let's assume that the buffer in router $(1;1)$ is full. Then, according to the routing policy of IPDeN 2.0 (see row 4 in Table 5.1), flit 1 of the green packet is deflected to port E of router $(1;1)$ and flit 1 of the red packet is buffered in the last slot of the buffer (see Figure 5.5b). In the next clock cycle, flit 2 of p can also conflict with flit 2 of the green packet over the port S of router $(1;1)$. Since the buffer in router $(1;1)$ remains full, flit 2 of the green packet is deflected and flit 2 of p is buffered in the last slot of the buffer (see Figure 5.5c and row 4 in Table 5.1). Two hops later, flit 1 of p reaches router R_d at the same time as flit 1 of the green packet (see Figures 5.5d and 5.5e). Flit 2 of p and green packet reach then R_d in the next clock cycle (see Figure 5.5f). Thus, flits arrive at their destination in the same order in which they were injected into the network.

Thus, IPDeN 2.0 is effectively an improved version of IPDeN NoC.

5.3 Worst-Case Communication Time Analysis

In this section, we analyze the WCCT between two PEs connected to IPDeN 2.0. The WCIT depends on the load of the network, therefore it can be determined as in Section 3.4.3. We discuss how to compute the WCTT in the rest of this section.

As explained in Section 5.2.2, in IPDeN 2.0 a flit may be buffered or deflected in routers in which it requests the port S . The WCTT of a flit traversing IPDeN 2.0 is then defined by the sum of the worst-case traversal time assuming zero contention and the total additional delay suffered by the flit due to both deflections and buffering.

In Section 4.4, we showed that $h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) + 2$ is the maximum number of hops a flit does from its origin PE to its destination PE if it suffers no deflection and is never buffered while traversing the network.

From Table 5.1 we get that a flit can suffer a deflection or a buffering in a router in which it requests the port S . It can then be deflected or buffered at most in all routers it reaches on its route along bypasses, i.e., $h_b^i(x_{q,i}^d, y_{q,i}^d)$ times. As we discuss in Section 5.2.2, it was also proven in Section 4.4 that the additional cost (in terms of clock cycles) added by each deflection is $S_x - 1$. See the formal proof of this statement in Lemma 23.

Lemma 23. The cost of a deflection is upper bounded by $S_x - 1$.

Proof. When a flit is deflected in a router, it must do S_x hops instead of 1, before reaching the same router as it would have if it could have used the S port instead. Therefore, each deflection involves an additional cost of $S_x - 1$ hops. $\square$

Lemma 24 then proves that a flit in a router can be buffered at most $S_x - 1$ cycles.

Lemma 24. The maximum cost of buffering a flit in a router is $S_x - 1$.

Proof. The buffer size in routers of IPDeN 2.0 is $S_x - 1$. A flit requesting port S of a router can be buffered only if there is an empty slot for it in the buffer. Then, the flit experiences the longest delay if it is buffered in the last slot of the buffer, i.e., in slot $S_x - 1$. Hence, the cost of buffering is *at most* $S_x - 1$. $\square$

Thus, it follows that the maximum cost of a deflection or buffering is upper bounded by $S_x - 1$. Then, the maximum total delay added by deflections and buffering is $h_b^i(x_{q,i}^d, y_{q,i}^d) \times (S_x - 1)$.

Combining the results above, we get that the WCTT of a flit of a packet of flow $f_{q,i}$ is given by

$$wctt_i = h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) + 2 + h_b^i(x_{q,i}^d, y_{q,i}^d) \times (S_x - 1)$$

It was proven in Section 4.4 that the WCTT of a flit of a packet of flow $f_{q,i}$ traveling IPDeN NoC is computed as

$$wctt_i = h_r^i(x_{q,i}^d, y_{q,i}^d) + h_b^i(x_{q,i}^d, y_{q,i}^d) \times (1 + (S_x - 1)) + 2 \quad (5.1)$$

Then, corollary 1 is a direct consequence of the discussion carried out in this section.

Corollary 1. The upper-bounds on the WCTT of flits are the same in IPDeN and IPDeN 2.0.

5.4 Experimental Results

5.4.1 Implementation

We implemented IPDeN 2.0 using the Verilog hardware description language. We synthesized a single router of IPDeN 2.0 for flits of 64 bits. It required 800 LUTs and 712 Flip-Flops (FFs).

The target platform was a Xilinx Kintex-7 FPGA. We compared IPDeN 2.0 with HopliteBuf [28], IPDeN, and three VC-based NoCs: CONNECT [67], IDAMC [88], and ProNoC [59] in terms of hardware resource utilization (see Table 5.2).

As reported in Table 5.2, to implement an 8x8 VC-based NoC are needed between 80000 and 100000 LUTs. Conversely, an 8x8 IPDeN 2.0 NoC requires ≈ 51000 LUTs, i.e., between 11% to 78% of the resources. Therefore, IPDeN 2.0 requires about half the hardware resources of the VCs-based NoC. A single HopliteBuf router requires 402 LUTs and an 8x8 HopliteBuf NoC requires ≈ 26000 LUTs, i.e., between 5% and 40% of the resources. This makes HopliteBuf $\approx 50\%$ less expensive than IPDeN 2.0 (in terms of LUT consumption). Nonetheless, IPDeN 2.0 uses smaller buffers, which is an advantage in terms of silicon footprints and power consumption. The size of IPDeN 2.0's buffers is also fixed, while those of HopliteBuf depend on the properties of the application using the network. Similarly, IPDeN 2.0 needs almost two times more resources than IPDeN due to its more complex routing policy. However, while flits in IPDeN are always deflected to alternative output ports if they contend with other flits, this is not the case in IPDeN 2.0. Therefore, as we showed in Section 5.4.2 IPDeN 2.0 has the clear advantage of reducing the flits average traversal time without impacting their worst-case communication times.

Table 5.2: Resources utilization of different NoCs in Kintex-7 FPGAs.

NoC	LUTs	% Resource utilization of the platform
8x8 ProNoC	100000	20%-150%
8x8 IDAMC	83000	18%-127%
8x8 CONNECT	96000	20%-147%
8x8 HopliteBuf	26000	5%-40%
8x8 IPDeN	30000	6%- 46%
8x8 IPDeN 2.0	51000	11%- 78%

5.4.2 Simulation Results

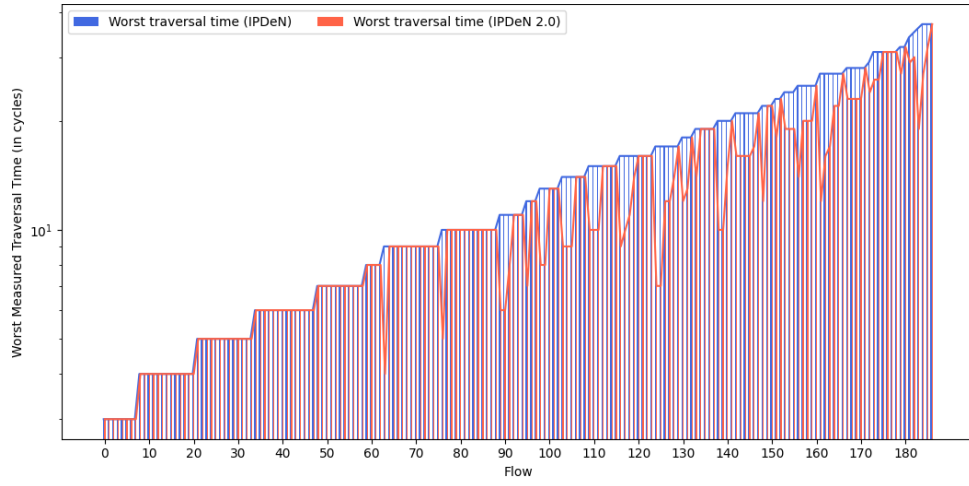
We use the synthetic test case proposed in [86] and the real case study involving Orion Crew Exploration Vehicle [71] to evaluate the performance of IPDeN 2.0 against IPDeN, HopliteBuf, and a generic VCs-based NoC while supporting realistic communication loads. These two test cases are commonly used to evaluate Ethernet-based systems. However, we have adapted them to be suitable for our experiments. We assume that the entire system resides on a single chip and that the processing elements communicate over NoCs instead of Ethernet networks.

Table 5.3 shows the configuration details for both test cases. The parameters of the traffic flows were set considering the parameters in [96] and [85]. The communication flows were generated according to a random traffic pattern. That is, the source and destination PEs of each flow were randomly generated. The packet size and inter-arrival time for each flow were randomly chosen within the intervals $[S_{min}, S_{max}]$ and $[T_{min}, T_{max}]$ (see Table 5.3), respectively. The flit size was set to 64 bits for both test cases.

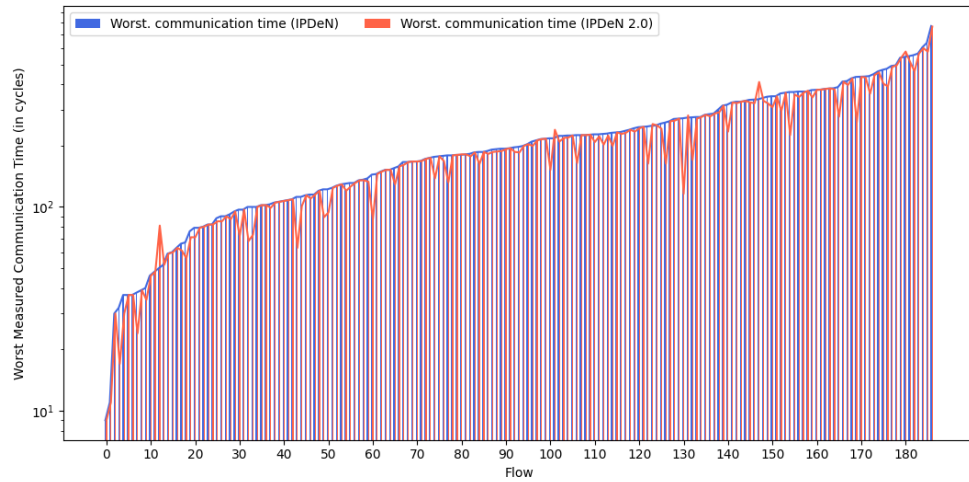
We implemented a network simulator to measure the traversal and communication times of flits that traverse IPDeN 2.0, IPDeN, HopliteBuf, and a VCs-based NoC. We then determined the worst *measured* traversal time (WMTT) and worst *measured* communication time (WMCT) for each flow and calculated the average *measured* traversal time (AMTT) and average *measured* communication time (AMCT).

Table 5.3: Configuration details for the test cases.

Test case	Network size	Number of PEs	Number of PEs generating communication flows	Number of flows	S_{min} (Bytes)	S_{max} (Bytes)	T_{min} (Clock cycles)	T_{max} (Clock cycles)
Synthetic	4x4	16	12	46	20	1443	2000	25000
Orion	6x6	36	31	187	20	1460	400	37500

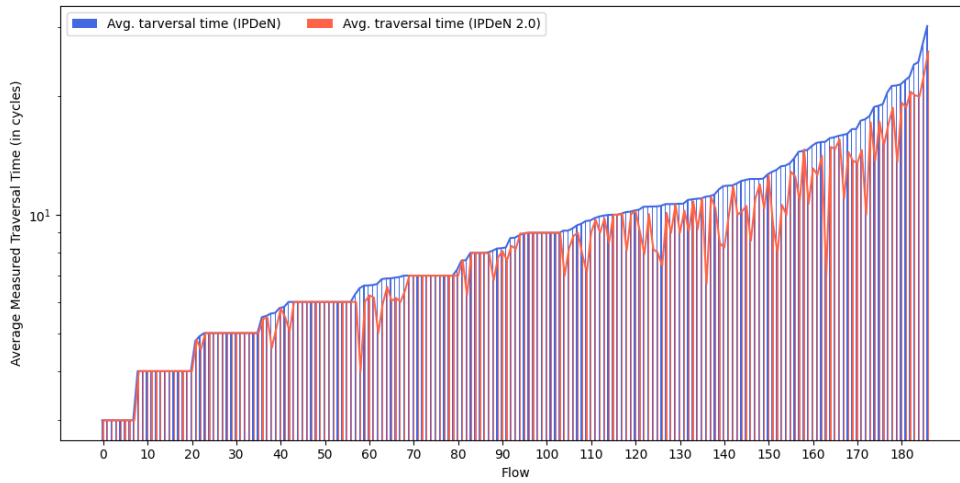


(a) Worst measured traversal time.

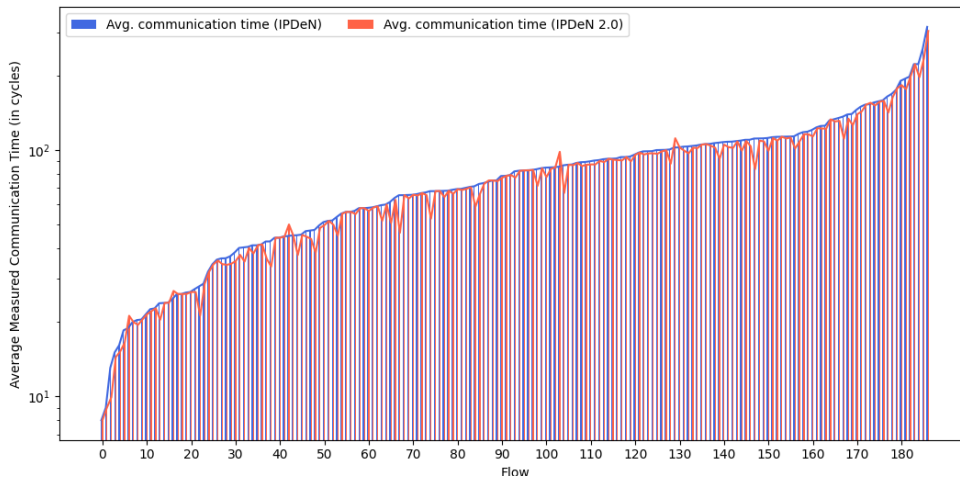


(b) Worst measured communication time.

Figure 5.6: IPDeN 2.0 vs IPDeN (Orion test case).



(c) Average measured traversal time.



(d) Average measured communication time.

Figure 5.6: IPDeN 2.0 vs IPDeN (Orion test case) (cont.).

Table 5.4: IPDeN 2.0 vs IPDeN (Orion test case).

	IPDeN 2.0<IPDeN	IPDeN 2.0=IPDeN	IPDeN 2.0>IPDeN
WMTT	61 out of 187 (32.6%)	126 out of 187 (67.4%)	0 out of 187 (0%)
WMCT	94 out of 187 (50.3%)	85 out of 187 (45.5%)	8 out of 187 (4.3%)
AMTT	109 out of 187 (58.3%)	74 out of 187 (39.6%)	4 out of 187 (2.1%)
AMCT	131 out of 187 (70.1%)	46 out of 187 (24.6%)	10 out of 187 (5.3%)

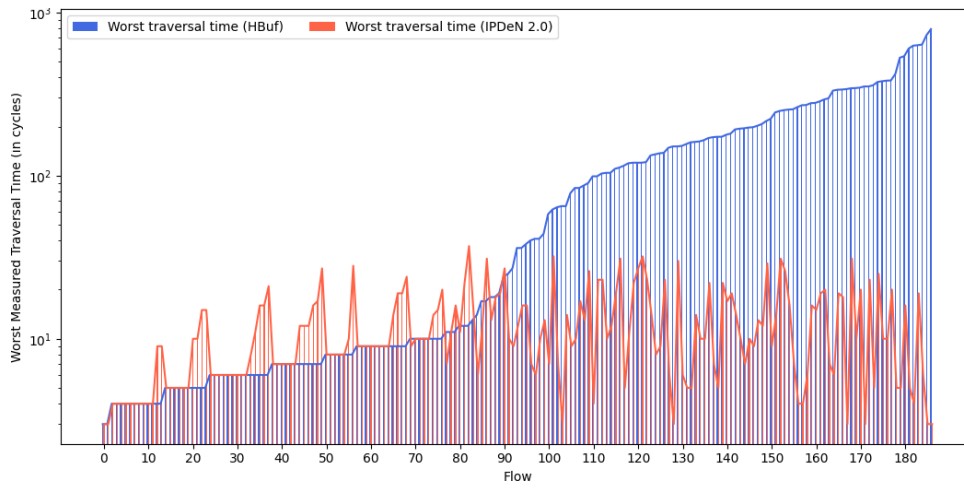
Figure 5.6 shows the WMTT, WMCT, AMTT, and AMCT for the Orion test case when flows traverse IPDeN 2.0 and IPDeN NoCs. Table 5.4 shows the number of flows for which IPDeN 2.0 performs better, the same, or worse than IPDeN. Notably, the average traversal time improves for

58.3% of the flows with the routing policy of IPDeN 2.0 whilst only 4 out of 187 flows see their average measured traversal time degraded in comparison to IPDeN. Interestingly, almost a third of the flows see their worst measured traversal time improved, and none see it degraded with IPDeN 2.0 in comparison to IPDeN. Similarly, the flows' communication times (i.e., considering the time flits must wait before being injected in the network) is significantly improving with IPDeN 2.0.

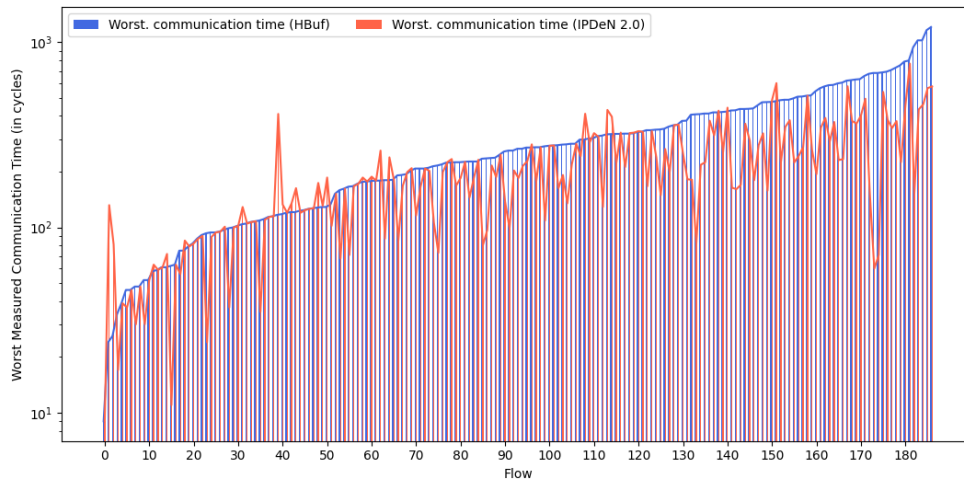
We evaluate IPDeN 2.0 against a HopliteBuf NoC with buffers of the same size as in IPDeN 2.0, i.e. buffers with 5 slots for the Orion test case. As we show in Table 5.5, IPDeN 2.0 performs better than a HopliteBuf NoC with 5-deep buffers in terms of worst measured traversal time and worst measured communication time for almost half of the communication flows, while the performance of less than 26% of the flows is worse with IPDeN 2.0. In terms of average time, no NoC solution dominates the other, as the traversal and communication times of nearly 35% of flows improve with IPDeN 2.0, while approximately another third of flows perform better with HopliteBuf. Note that 20405 out of 202039 flits never reach their destination when they travel through a 5-deep HopliteBuf because the buffers in the routers overflow (see Table 5.7). For HopliteBuf, we show in Table 5.8 the maximum number of flits lost by a flow, the average number of flits lost per flow, and the number of flows having lost flits. We then increased the buffer size in HopliteBuf to 128 slots. Figure 5.7 shows the WMTT, WMCT, AMTT, and AMCT for IPDeN 2.0 and a HopliteBuf with 128-deep buffers. Table 5.6 shows the number of flows for which IPDeN 2.0 performs better, the same, or worse than the 128-deep HopliteBuf. Even with large buffers for HopliteBuf, the scales clearly tilt in favor of IPDeN 2.0 in terms of average and worst traversal and communication times. Furthermore, as shown in Table 5.7, there are still 1653 flits that never reach their destination even when each router contains a 128-deep buffer. Therefore, to support realistic communication loads, HopliteBuf's buffers must be oversized. As discussed in 4.6.1, the size of the buffers in HopliteBuf depends on the set of communication flows transmitted through the NoC. However, for the Orion test case, the routers in IPDeN 2.0 only need a single 5-deep buffer to ensure good behavior, since the buffer size depends only on the network topology.

Table 5.5: IPDeN 2.0 vs 5-deep HopliteBuf (Orion test case).

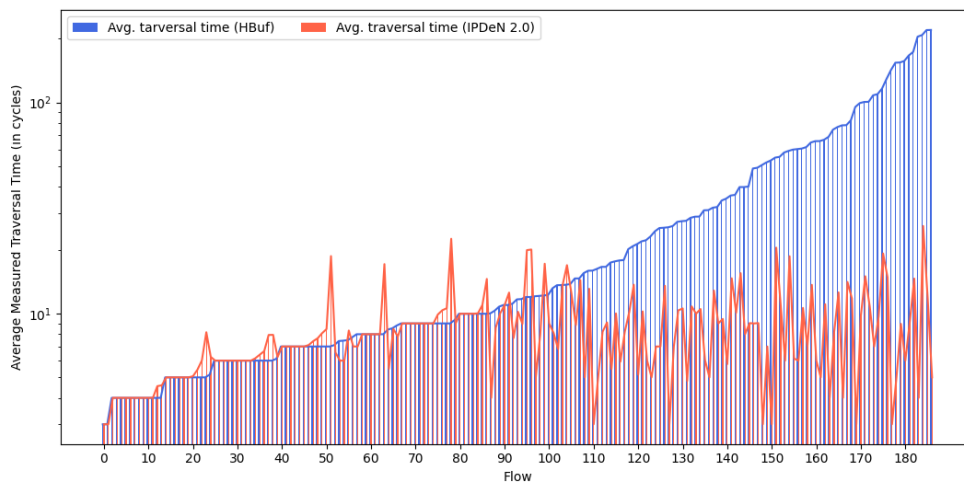
	IPDeN 2.0<HopliteBuf	IPDeN 2.0=HopliteBuf	IPDeN 2.0>HopliteBuf
WMTT	91 out of 187 (48.7%)	61 out of 187 (32.6%)	35 out of 187 (18.7%)
WMCT	89 out of 187 (47.6%)	50 out of 187 (26.7%)	48 out of 187 (25.7%)
AMTT	65 out of 187 (34.8%)	58 out of 187 (31%)	64 out of 187 (34.2%)
AMCT	61 out of 187 (32.6%)	44 out of 187 (23.5%)	82 out of 187 (43.9%)



(a) Worst measured traversal time.

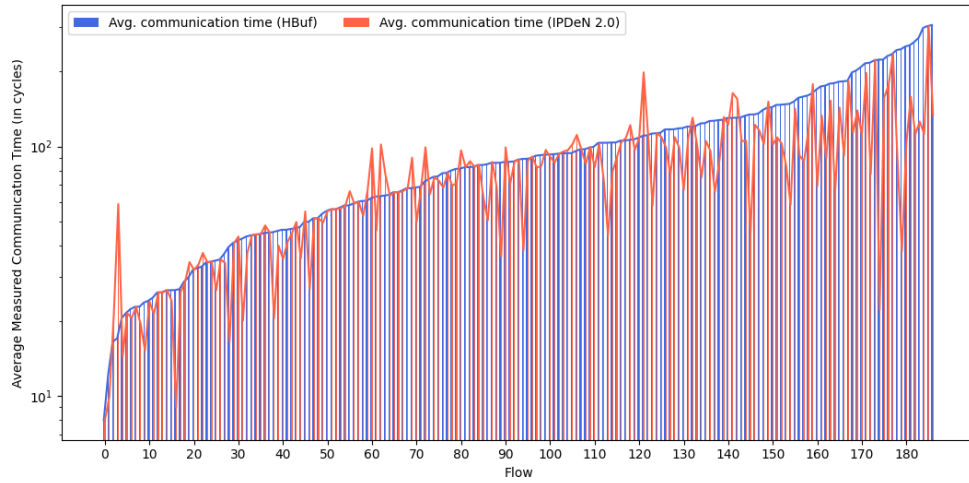


(b) Worst measured communication time.



(c) Average measured traversal time.

Figure 5.7: IPDeN 2.0 vs 128-deep HopliteBuf (Orion test case).



(d) Average measured communication time.

Figure 5.7: IPDeN 2.0 vs 128-deep HopliteBuf (Orion test case) (cont.).

Table 5.6: IPDeN 2.0 vs 128-deep HopliteBuf (Orion test case).

	IPDeN 2.0<HopliteBuf	IPDeN 2.0=HopliteBuf	IPDeN 2.0>HopliteBuf
WMTT	102 out of 187 (54.5%)	53 out of 187 (28.3%)	32 out of 187 (17.1%)
WMCT	110 out of 187 (58.8%)	41 out of 187 (21.9%)	36 out of 187 (19.3%)
AMTT	103 out of 187 (55.1%)	51 out of 187 (27.3%)	33 out of 187 (17.6%)
AMCT	113 out of 187 (60.4%)	35 out of 187 (18.7%)	39 out of 187 (20.9%)

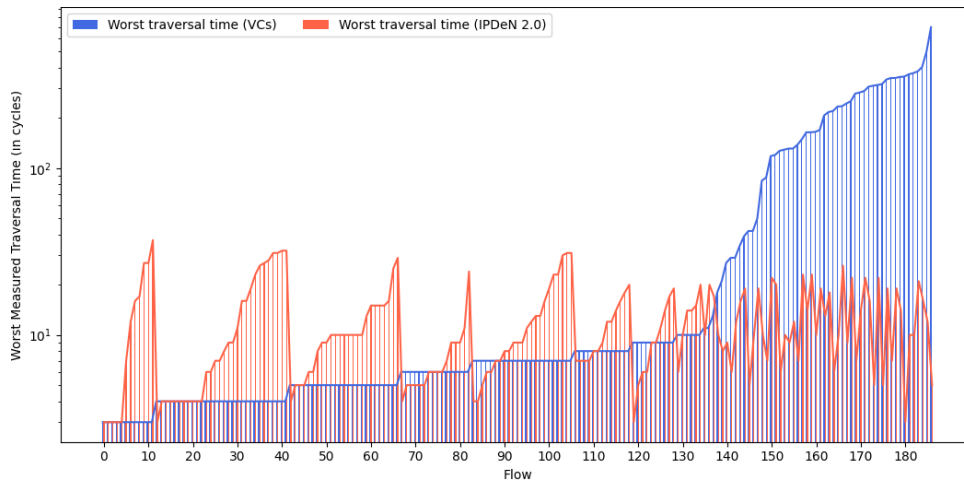
Table 5.7: Number of lost flits (Orion test case).

	IPDeN 2.0	HopliteBuf	
		5-deep	128-deep
Lost flits	0	20405	1653

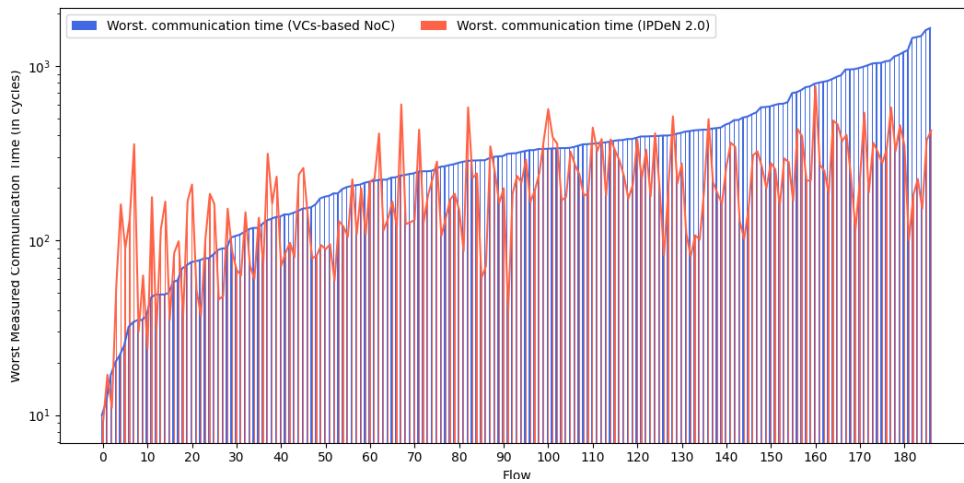
Table 5.8: Analysis of the lost flits in HopliteBuf (Orion).

	Max. number of flits lost by a flow	Avg. number of flits lost per flow	Number of flows having lost flits
5-deep HopliteBuf	3212	215	95
128-deep HopliteBuf	564	79	21

In Figure 5.8, we evaluate IPDeN 2.0 against the VCs-based NoC. In Table 5.9, we show that for the Orion test case, $\approx 77\%$ of flows experience a significant reduction in their worst communication time when traversing IPDeN 2.0. The remaining 23% of flows see their worst communication time reduced in the VCs-based NoC. IPDeN 2.0 also performs significantly better than the VCs-based NoC in terms of average communication time for about 90% of the flows. Only 20 flows see their average communication time degraded with IPDeN 2.0. In terms of traversal time, however, the VCs-based NoC performs better than IPDeN 2.0 for approximately half of the flows.

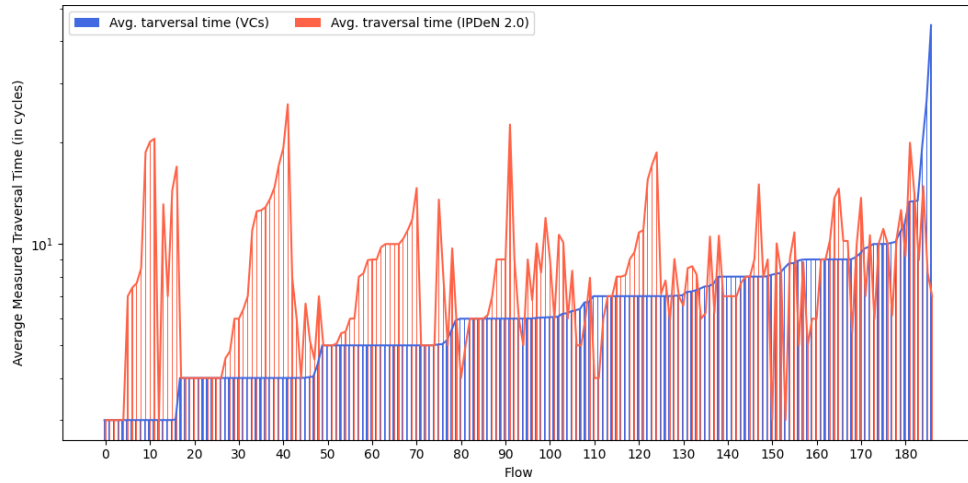


(a) Worst measured traversal time.

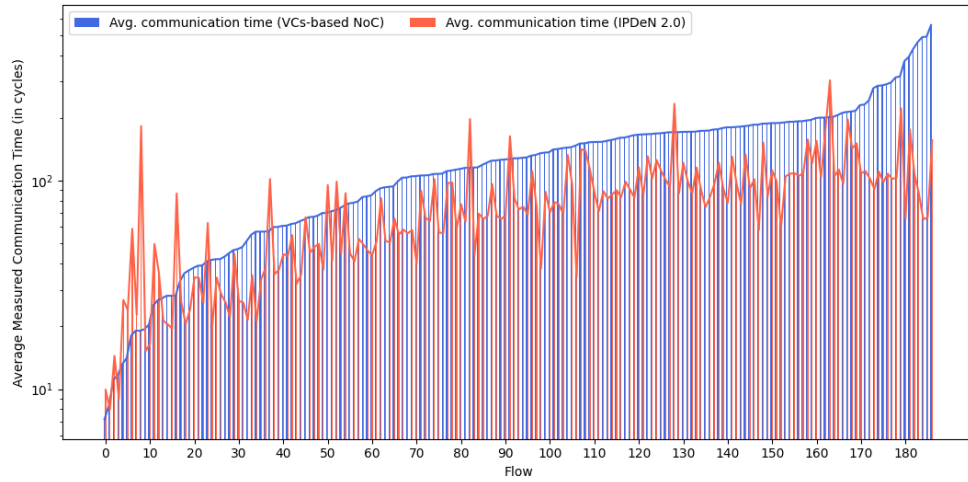


(b) Worst measured communication time.

Figure 5.8: IPDeN 2.0 vs VCs (Orion test case).



(c) Average measured traversal time.



(d) Average measured communication time.

Figure 5.8: IPDeN 2.0 vs VCs (Orion test case) (cont.).

Table 5.9: IPDeN 2.0 vs VCs-based NoC (Orion test case).

	IPDeN 2.0<VCs-NoC	IPDeN 2.0=VCs-NoC	IPDeN 2.0>VCs-NoC
WMTT	72 out of 187 (38.5%)	29 out of 187 (15.5%)	86 out of 187 (46%)
WMCT	143 out of 187 (76.5%)	0 out of 187 (0%)	44 out of 187 (23.5%)
AMTT	49 out of 187 (26.2%)	28 out of 187 (15%)	110 out of 187 (58.8%)
AMCT	167 out of 187 (89.3%)	0 out of 187 (0%)	20 out of 187 (10.7%)

Tables 5.10, 5.11, and 5.12 show the number of flows for which IPDeN 2.0 performs better, the same, or worse than IPDeN, HopliteBuf, and the VCs-based NoC for the synthetic test, respectively. The plots can be found in an Appendix C.

The average gain and loss (in %) per group of flows on the WMTT, WMCT, AMTT, and AMCT when comparing IPDeN 2.0 with IPDeN, 128-deep HopliteBuf, and the VC-based NoC for the Orion test case can be found in an Appendix C.

Table 5.10: IPDeN 2.0 vs IPDeN (Synthetic test case).

	IPDeN 2.0<IPDeN	IPDeN 2.0=IPDeN	IPDeN 2.0>IPDeN
WMTT	5 out of 187 (10.9%)	40 out of 187 (87%)	1 out of 187 (2.2%)
WMCT	31 out of 187 (67.4%)	5 out of 187 (10.9%)	10 out of 187 (21.7%)
AMTT	22 out of 187 (47.8%)	14 out of 187 (30.4%)	10 out of 187 (21.7%)
AMCT	29 out of 187 (63%)	0 out of 187 (0%)	17 out of 187 (37%)

Table 5.11: IPDeN 2.0 vs 128-deep HopliteBuf (Synthetic test case).

	IPDeN 2.0<HopliteBuf	IPDeN 2.0=HopliteBuf	IPDeN 2.0>HopliteBuf
WMTT	35 out of 187 (76.1%)	1 out of 187 (2.2%)	10 out of 187 (21.7%)
WMCT	41 out of 187 (89.1%)	0 out of 187 (0%)	5 out of 187 (10.9%)
AMTT	35 out of 187 (76.1%)	1 out of 187 (2.2%)	10 out of 187 (21.7%)
AMCT	39 out of 187 (84.8%)	0 out of 187 (0%)	7 out of 187 (15.2%)

Table 5.12: IPDeN 2.0 vs VCs-based NoC (Synthetic test case).

	IPDeN 2.0<VCs-NoC	IPDeN 2.0=VCs-NoC	IPDeN 2.0>VCs-NoC
WMTT	19 out of 187 (41.3%)	5 out of 187 (10.9%)	22 out of 187 (47.8%)
WMCT	25 out of 187 (54.3%)	0 out of 187 (0%)	21 out of 187 (45.7%)
AMTT	12 out of 187 (26.1%)	5 out of 187 (10.9%)	29 out of 187 (63%)
AMCT	28 out of 187 (60.9%)	0 out of 187 (0%)	18 out of 187 (39%)

5.5 Chapter Summary

In this chapter, we presented IPDeN 2.0, an improved version of IPDeN with a routing policy based on selective deflection and buffering of flits to enhance their traversal time while maintaining bounded the WCTT. The solution preserves the in-order flit delivery property introduced in IPDeN. We demonstrated that, similar to IPDeN, the timing behavior in IPDeN 2.0 is deterministic, and the WCTT of flits depends solely on the properties of the communication flow under analysis. We proved that the upper-bounds on the WCTT of flits are the same in IPDeN 2.0 as in IPDeN. Experimental results shown that IPDeN 2.0 outperforms IPDeN and HopliteBuf in terms of actual traversal and communication times, without affecting the upper-bounds on worst-case traversal and communication times. Experimental results shown that IPDeN 2.0 outperforms IPDeN and HopliteBuf in terms of actual traversal and communication times.

Chapter 6

Traffic Injection Regulation Protocol

As mentioned earlier, NoCs consist of numerous shared resources, such as routers and links, that may be susceptible to contention scenarios. These scenarios, in turn, can potentially lead to issues like packet collisions, packet transmission delays, and even critical problems like deadlocks, live-locks, or packet loss due to buffer overflows if not managed effectively. While many contention-aware approaches primarily address scenarios where multiple flits traveling through the network attempt to use the same communication channel at the same time, contention scenarios may also occur when the volume of data to be transmitted exceeds the network's bandwidth or when flits already traveling through the network block the injection of new flits by PEs. Such contention scenarios frequently occur when the number of communication flows mapped on each PE increases and/or when the number of packets generated in each time interval increases. In these cases, the injection of new flits may remain blocked for long periods of time until the requested output ports become available.

HopliteRT*, IPDeN, and IPDeN 2.0 assume that there is no regulation of traffic injection on PEs, i.e., PEs can inject flits as fast as possible. As a direct consequence of this best-effort access strategy, communication flows mapped on PEs located in regions of low congestion can use network bandwidth indiscriminately and therefore have very low injection delay. Those PEs may then "overuse unfairly" the network bandwidth, thereby resulting in generating lots of contention for other PEs downstream, eventually preventing those other PEs from injecting flits altogether. The communication time of packets originating from some PEs may thus increase excessively. See Section 3.3.1 for an example of long blocking suffered by PEs in HopliteRT*.

If traffic injection at each PE is not regulated, either locally or globally, it may lead to certain PEs experiencing starvation. Furthermore, it makes the system vulnerable to attacks that aim to disrupt or temporarily suspend its normal functioning, e.g., Denial-of-Service attacks. On the other hand, long transmission latency is a major concern in real-time NoCs, where packets must reach their destinations within predefined deadlines [39]. To address this challenge when designing solutions for real-time applications, it is essential to integrate a traffic injection regulation protocol into NoCs to better distribute network bandwidth and therefore ensure different QoS levels to different traffic classes.

In this chapter, we present the D-TSR protocol, a new strategy to regulate traffic injection in NoCs based on dynamic time slots requests by PEs that are unable to inject their flits into the network. Those requester PE send requests to PEs located upstream in the network to free some bandwidth. That is, it prevents some PEs from being blocked indefinitely due to unfair bandwidth usage by other PEs.

The chapter is structured as follows. Section 6.1 provides an overview of related work available in the literature. The system model is presented in Section 6.2. Section 6.3 presents our main contribution and Section 6.4 derives a new upper-bound on the WCIT of flits when traffic injection is regulated by the D-TSR protocol. Finally, several experimental results are discussed in Section 6.5.

The content of this chapter was previously published in the following paper:

- **Y. Ribot González**, G. Nelissen, and E. Tovar. Traffic Injection Regulation Protocol based on free time-slots requests. In 29th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'23), 2023.

6.1 Related Work

Over the years, several packet injection rate regulation techniques have been proposed in lieu of free-to-go admission to regulate the number of packets traversing the network at the same time. [65] proposes a predictive closed-loop on/off flow control mechanism. [64] presents a flow control mechanism based on adaptive input rate control for NoCs with wormhole routing and virtual channels. In [47], Jafari et al. introduce a flow control scheme that admits Max-Min fairness criterion. However, these solutions do not take into account the presence of sporadic bursts of traffic that occur when routes merge and cause temporary saturation of router buffers. Therefore, they cannot completely avoid congestion and instead rely on back-pressure mechanisms to compensate for it.

Network Calculus is a mathematical theory that provides a set of mathematical tools and techniques for analyzing and evaluating the performance of computer networks [16], [52]. It allows for formalizing the presence of sporadic traffic bursts. The commonly used (σ, ρ) -traffic scheme was originally developed in network calculus. [55], [46], [45], [19] propose static (σ, ρ) -based traffic regularization methods. Such solutions guarantee a deterministic worst-case delay bound through a formal analysis but require prior knowledge of traffic and cannot adapt to the dynamic behavior of a typical complex system, leading to over- and under-regulation. [24], [56] then proposes dynamic traffic regulation schemes. However, they do not address the fault tolerance of data and some traffic data can be approximated rather than transmitted.

Some other works, e.g., [90] and [6], rely on leaky buckets to regulate the network bandwidth distribution. The approach allows to limit the number of packets PEs can send in bursts when output ports are available. This prevents some PEs from being blocked indefinitely due to unfair bandwidth usage by other PEs. However, it rarely use runtime information to make decisions about which PEs can inject flits into the network.

[11] proposes an approximation design to control the volume of traffic injection data using network congestion information. However, it ignores the impact of different traffic flows on network congestion. Moreover, in each router, a congested packet must wait until the previous packet has fully traversed the network before being transmitted. Congestion then causes a long delay in traffic. In [89], an approximation-based dynamic traffic regulation is introduced to address traffic congestion by using value prediction mechanisms to reduce the amount of data injected into the network.

6.2 System Model

We adopt a system model identical to that of HopliteRT* (refer to Section 3.1)¹.

6.3 Our Solution

According to HopliteRT*'s routing policy¹, a PE π_q is prevented from injecting a new flit to port E of its associated router R_q if: (1) a flit comes from the router directly west of R_q (i.e., router R_{q^w}) and requests port E of R_q (see rows number 1 and 4 in Table 3.1); (2) a flit is deflected to port E of R_q after conflicting with another flit for its port S (i.e., when a flit entering via port N competes with a flit coming from R_{q^w} for port S of R_q) (see row 2 in Table 3.1); or (3) there is a flit entering R_q via port W requesting port S since HopliteRT*'s routing policy does not support this scenario (see row 5 in Table 3.1). Therefore, π_q is able to inject any pending flit on port E of R_q as soon as R_{q^w} does not send a flit towards R_q .

To prevent PEs from blocking endlessly, we propose to implement a traffic injection regulator that makes PEs *collaborate* so that R_{q^w} does not constantly use port W of R_q when R_q requires to access the network. The approach we proposed is called D-TSR and works as follows.

Under D-TSR, PE π_q connected to router R_q may ask routers that are directly west for clock cycles in which they refrain from forwarding flits toward R_q . The free slots generated that way can then be used to inject π_q 's own flits in the network. That is, a PE π_q that cannot inject its flits asks the router R_{q^w} for free-clock slots. To satisfy π_q 's request, the PE π_{q^w} (i.e., the PE connected to router R_{q^w}) stops injecting flits to port E of R_{q^w} during a certain number of clock cycles. Specifically, each PE π_{q^w} has two variables D_{q^w} and G_{q^w} , defined as the number of clock cycles D_{q^w} PE π_{q^w} uses to inject its own flits out of G_{q^w} free-clock cycles it gets when it receives requests for free-slots from π_q . Note that, the number of flits π_{q^w} can inject is regulated by D_{q^w} and G_{q^w} , while π_q injects flits as fast as possible regardless of the values of D_q and G_q if the PE directly east to π_q does not request free-slots. For example, if $D_{q^w} = 3$ and $G_{q^w} = 10$, π_{q^w} uses 3 out of 10 clock cycles it gets, and π_q uses the remaining 7 clock cycles. The following property holds for D-TSR protocol.

¹We use HopliteRT* to present the D-TSR protocol, but it also works with IPDeN and IPDeN 2.0 in a strictly identical manner. The injection protocol is independent of the routing policy implemented in the router.

Property 3. If the PE directly east to π_q requests free slots, then the number of flits PE π_q injects in a given time interval is regulated by the variables D_q and G_q . Otherwise, π_q injects flits as fast as possible.

However, even if PE π_{q^w} satisfies the request of PE π_q , a flit arriving in R_{q^w} via port N or W may be routed to its port E , thus interfering with flit injection into R_q . In this case, R_{q^w} asks its directly west router not to inject flit in the next clock cycle. This router in turn does exactly the same if it cannot assure π_{q^w} a free-time slot due to incoming traffic from upstream routers, and so on until a time slot is enabled for π_q to use. Note that, it is possible that π_q requests free-clock cycles to π_{q^w} , but π_{q^w} was not able to inject its own flits during the previous cycles and therefore also requests free-time slots. In this case, π_{q^w} will reserve some of the received slots for itself and give the rest to π_q . That is, π_{q^w} will use D_{q^w} out of G_{q^w} free-clock cycles it receives and pass the remaining to its east PE, i.e., PE π_q . The behavior repeats until no more requests come from the east. Therefore, Property 4 below holds.

Property 4. The number of free-clock cycles π_q receives depends on the number of clock cycles the other PEs can offer.

Example. Consider a 4×4 HopliteRT* NoC and three routers R_p , R_m , and R_q with exactly the same properties as in the previous example. Suppose that $G_p = G_m = 10$, $D_p = 5$, and $D_m = 7$ (see Figure 6.1). Consider also the same scenarios analyzed. Assume that at the time π_p starts injecting the red packet at port E of R_p , no request for free clock cycles has yet been received from the PE located directly to the east, i.e., PE π_m (see Figure 6.1a). Thus, π_p injects the red packet as fast as possible (see Property 3). In the next clock cycle, a yellow packet releases within π_m , but cannot enter the network as red flits prevent π_m from injecting flits on port E of R_m . π_m then asks π_p not to send flits over port W of router R_m (see Figure 6.1b). To comply with π_m 's request, π_p uses some of the free clock cycles it receives to inject its own flits and forwards the rest to π_m . Specifically, since $D_p = 5$ and $G_p = 10$, π_p consumes 5 out of 10 free-clock cycles it gets and passes the other 5 free-slots to π_m (see from Figure 6.1b to Figure 6.1h). While π_p injects flits, a green packet releases within π_q (see Figure 6.1c). Similar to π_m , π_q cannot inject flits to port E of R_q since it is being used by flits coming from R_p . π_q then requests free-time slots from π_m . Once π_m starts receiving free-clock cycles, it takes the first 7 cycles for itself and offers the remaining 3 cycles to π_q (see from Figure 6.1d to Figure 6.1h). Note that, π_m does not inject the 7 flits consecutively. Since π_q supplies π_m with 5 out of 10 cycles, π_m injects 5 flits first and then 2 more flits.

A special case of what has been discussed so far is that π_q is prevented from injecting flits and therefore requests free-slots from π_{q^w} , but no packet has released within π_{q^w} . In this scenario, π_{q^w} forwards any free-clock cycles it receives to π_q .

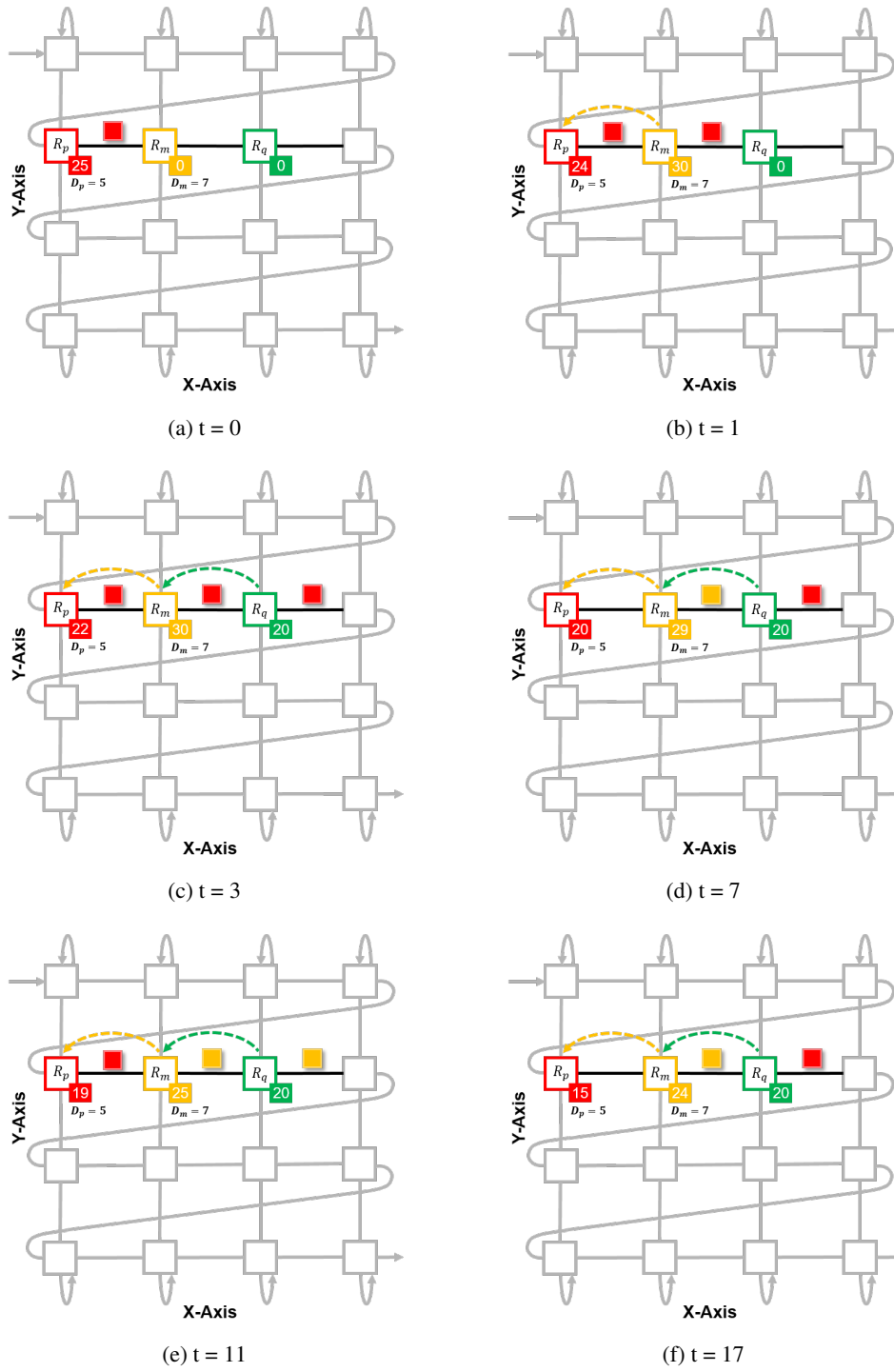


Figure 6.1: Example 1 of D-TSR protocol in HopliteRT* NoC.

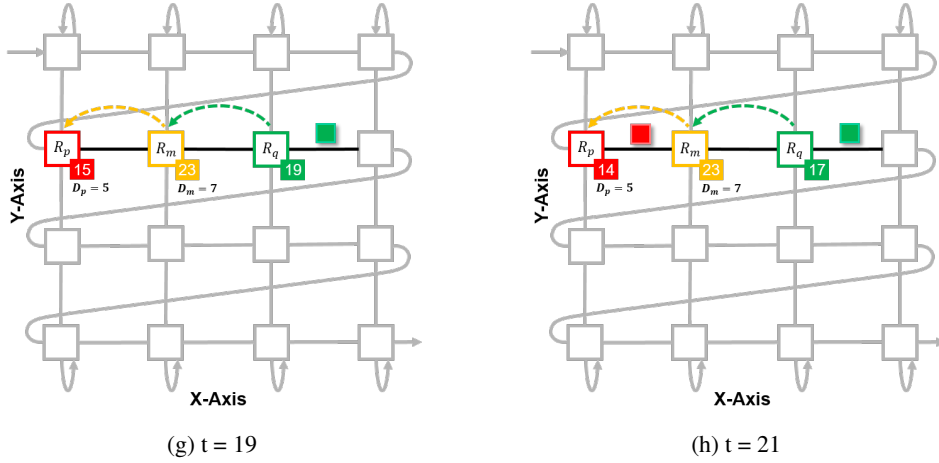


Figure 6.1: Example 1 of D-TSR protocol in HopliteRT* NoC (cont.).

Example. Consider a 4×4 HopliteRT* NoC with exactly the same properties as in the previous example. Also consider the same scenarios analyzed in Example 3 (see Figure 6.2). As in Example 3, π_p starts injecting the red packet as fast as possible. A few clock cycles later, a green packet releases within π_q but cannot enter the network as red flits prevent π_q from injecting flits on port E of R_q . π_q requests then π_m free-slots to inject the green flits (see Figure 6.2b). To meet π_q 's request, in the next clock cycle π_m asks π_p not to send flits over port W of router R_m (see Figure 6.2c). To comply with π_m 's request, π_p uses 5 out of 10 free-clock cycles it receives and passes the other 5 slots to π_m (see from Figure 6.2c to 6.2e). However, there is no flit in π_m to inject, so π_m forwards all slots it gets to π_q . Thus, π_p injects 5 flits before it is blocked again.

HopliteRT* is a deflection-based solution, but the deflection of flits does not affect our strategy to regulate traffic injection in PEs. In fact, the behavior is exactly the same as that discussed so far.

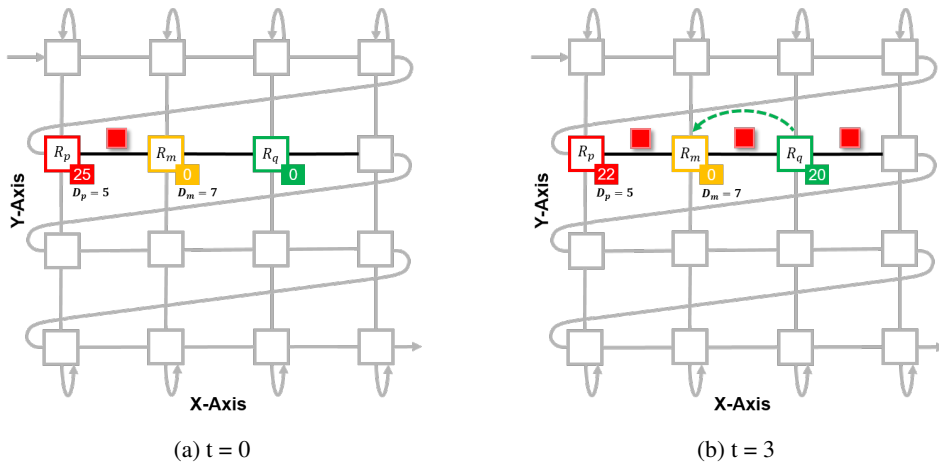


Figure 6.2: Example 2 of D-TSR protocol in HopliteRT* NoC.

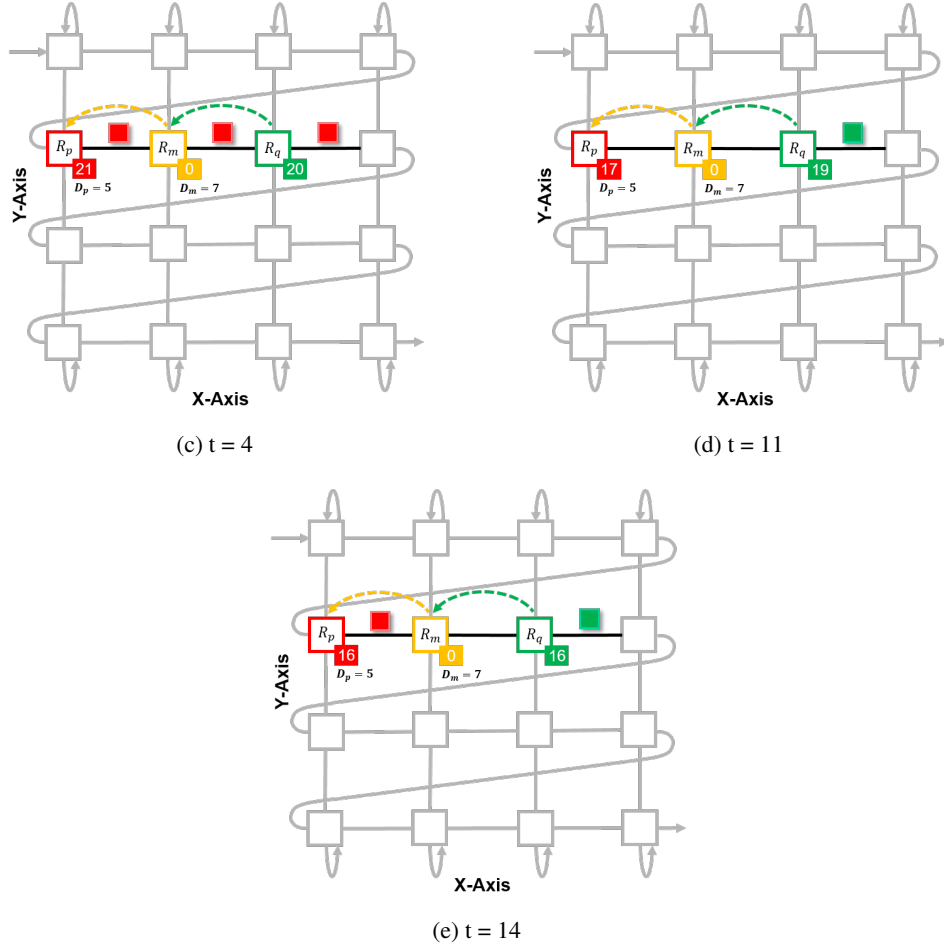


Figure 6.2: Example 2 of D-TSR protocol in HopliteRT* NoC (cont.).

Example. Consider a 4×4 HopliteRT* NoC. Consider also a red high-priority communication flow entering R_m via port W and conflicting for port S of R_m with a blue low-priority flow entering from port N . Assume $G_p = G_m = G_n = 10$, $D_p = 2$, $D_m = 7$, and $D_n = 4$ (see Figure 6.3a). In this contention scenario, red flits are routed through port S of R_m , while blue flits are deflected to the east (see Figure 6.3b). Suppose that at the time when red and blue flows traverse the network, a green packet releases within π_q with destination at coordinates (3;3). However, the injection of green flits is hindered by the blue flits that have been deflected in R_m and currently occupy port E of R_q . π_q then asks π_m not to send flits over port W of router R_m (see Figure 6.3c), and π_m in turn forwards this request to π_p (see Figure 6.3d). To comply π_m 's request, π_p regulates the number of flits it injects. That is, π_p uses 2 out of 10 free-clock cycles it gets and passes the remaining 8 free-slots to π_m (see from Figure 6.3d to 6.3f). π_m then forwards all slots it receives from π_p to π_q , since no yellow packet was released within π_m . Note that, π_n continues to inject flits. Since no red flit arrives at π_m , blue flits are no longer deflected, so π_q can inject flits (see Figures 6.3g and 6.3h). The fact that blue flits are not deflected for a few clock cycles (i.e., as long as π_p does not inject a flit) has a positive effect on average packets' traversal time.

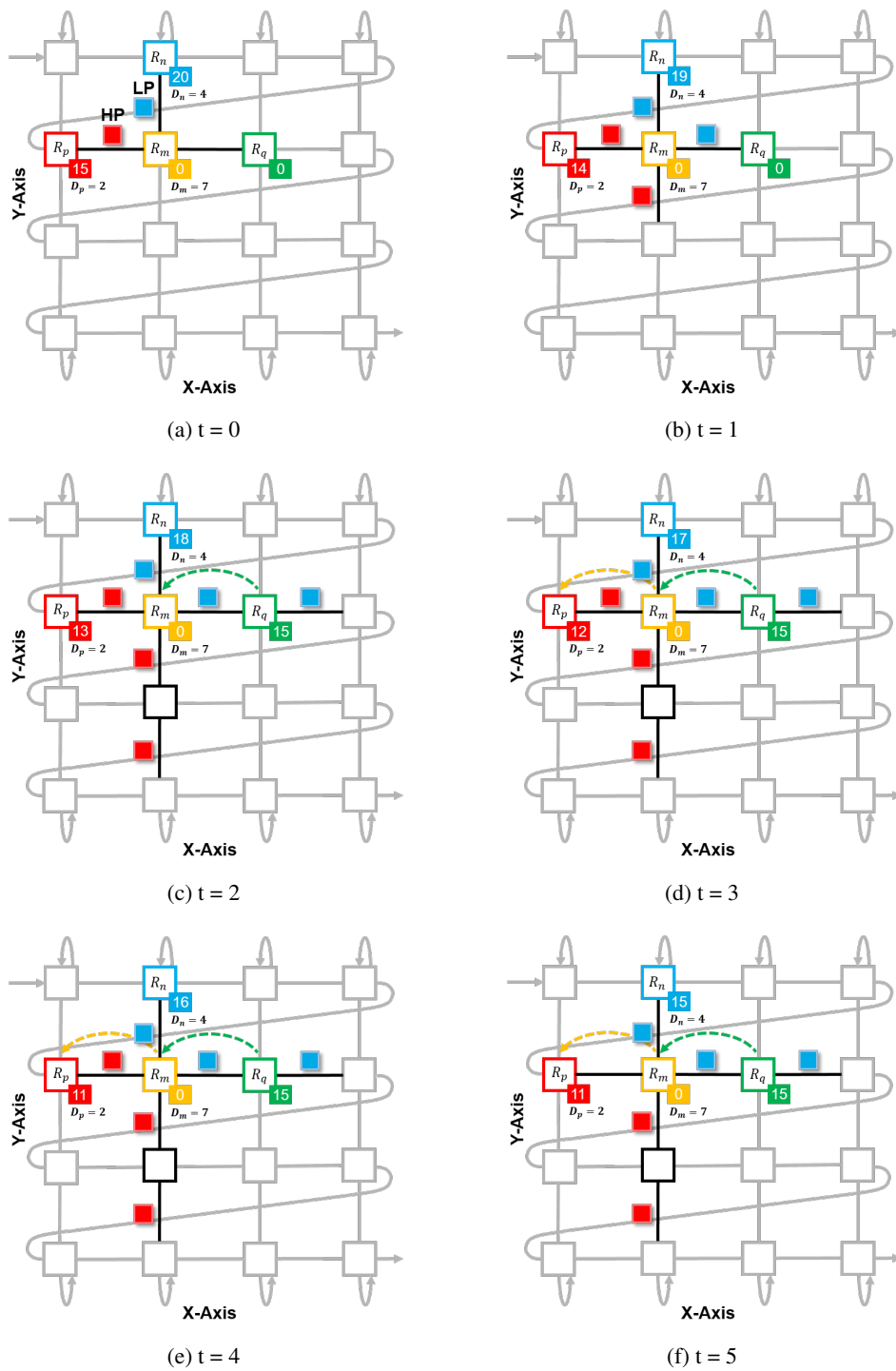


Figure 6.3: Example 3 of D-TSR protocol in HopliteRT* NoC.

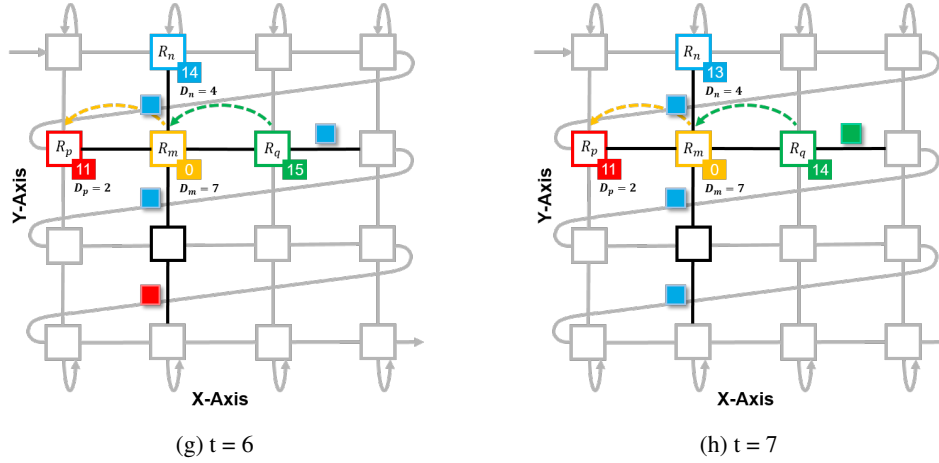


Figure 6.3: Example 3 of D-TSR protocol in HopliteRT* NoC (cont.).

6.3.1 Implementation

The proposed solution is implemented in hardware in the routers. Each pair of consecutive routers on the ring (i.e., on the horizontal dimension) is connected by an additional signal called y (red links in Figure 6.4). For example, routers R_{q-1} and R_q are connected by the signal $y_{q-1,q}$, and routers R_q and R_{q+1} are connected by the signal $y_{q,q+1}$. Specifically:

- ▷ Signal $y_{q-1,q}$ takes the value 1 if:
 - PE π_q wants to inject a flit but is blocked by traffic originating from R_{q-1} or if,
 - PE π_q receives a request for free-slots from R_{q+1} (the router directly east), i.e., if $y_{q,q+1} = 1$ and π_q forwards traffic originating from π_{q-1} towards π_{q+1} .
- ▷ In any other case, signal $y_{q-1,q}$ is set to 0.

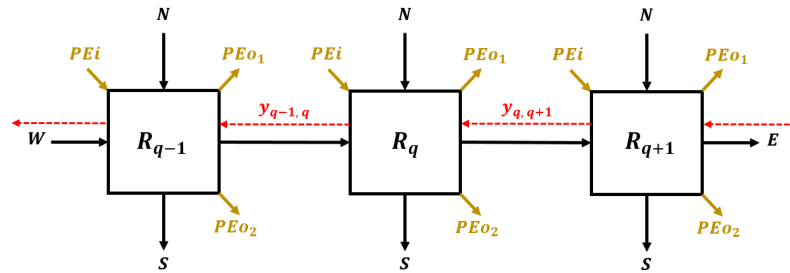


Figure 6.4: Inputs, outputs, and signals in HopliteRT* + D-TSR.

6.3.2 Discussion

In solutions such as HopliteRT*, where there is no traffic injection regulation and PEs have the lowest priority to use network resources, communication flows indiscriminately use network bandwidth without regard to the impact on network performance. That is, flows that are mapped in regions with lower congestion or that had the chance to access the network first will prevent other flows from injecting flits, leading to an increase on the worst-case communication time of the blocked flows. In Example 2, we can observe that the yellow flow can inject its packets only after the red flow has finished its injections. The same is true for the green flow, which can only access the network after the red and yellow flows have stopped injecting flits.

Our solution avoids this behavior. The approach allows to regulate the injection rate of PEs according to the network congestion situation, i.e., the injection of certain flits into the network can be delayed to distribute the network bandwidth fairly among all PEs. As a result, network links are used more effectively, which can lead to low traffic delays and improved overall performance. In Example 3, we observe that each communication flow has the possibility to inject flits. At the end of the example, the red, yellow, and green flows have injected 11, 6, and 3 flits, respectively. Note that, our solution does not impact the usable network bandwidth, but only distributes it more fairly according to a pre-chosen distribution defined by the variables D_q and G_q defined in each router. If 16 flits move from one node to another in each clock cycle, i.e., one flit per link. When our contribution is added to HopliteRT*, the network still supports 16 flits that can traverse the network instantaneously at each clock cycle in HopliteRT*, then 16 flits can traverse the network instantaneously at each clock cycle when our traffic injection protocol is added.

Finally, in applications where all information exchange over the network does not have the same level of importance and/or priority, our solution can play a major role by specifying the bandwidth each PE will use in each time interval. Then, PEs mapped with more important communication flows may be allocated more bandwidth by limiting the injection of PEs upstream more drastically.

6.4 Worst-Case Injection Time Analysis

In the previous section, we introduced a new protocol for regulating traffic injection into NoCs. As mentioned earlier, this new strategy regulates the number of flits injected by each PE depending on the level of congestion in the network. Hence, in this section, we derive a new upper-bound on the WCIT of flits when traffic injection is regulated by the D-TSR protocol. We assume that $T_{q,i} \geq C_{q,i}$. We also assume that each flow $f_{q,i}$ can have at most one packet within π_q waiting to be injected into the network at any time instant, i.e., $T_{q,i} \geq wcit_i$.

Lemma 25. Let $\hat{n}_{q^w}(t)$ be the maximum number of clock cycles during which flits enter router R_q via port W and conflict with the injection of flits by π_q in any time interval of length t . The WCIT of a flit of flow $f_{q,i}$ is a value of t that satisfies the following equation:

$$C_{q,i} = D_q \cdot \left\lfloor \frac{t - \hat{n}_{q^w}(t) - (G_q - D_q)}{G_q} \right\rfloor + \min(D_q, (t - \hat{n}_{q^w}(t) - (G_q - D_q)) \bmod G_q) \quad (6.1)$$

Proof. In the worst-case, the flit under analysis is the last flit of a packet of flow $f_{q,i}$. Consequently, it is injected into the network only after all the preceding flits of the same packet have been injected. Therefore, the WCIT of a flit of flow $f_{q,i}$ is determined by the number of clock cycles required for PE π_q to inject $C_{q,i}$ flits.

The minimum number of clock cycles required for π_q to inject $C_{q,i}$ flits is $C_{q,i}$ clock cycles, i.e., one clock cycle per flit. However, in NoC solutions like HopliteRT*, IPDeN, or IPDeN 2.0, where PEs always have the lowest priority when using ports, PEs can only inject flits if no flit from another PE requests the same output port at the same time. Consequently, π_q can solely inject flits during clock cycles when the requested output port is available.

From Property 4, we get that the number of flits π_q injects into the network depends on the number of free-slots forwarded to it by the PE directly west π_{q-1} . Let $\hat{n}_{q^w}(t)$ be the maximum number of clock cycles during which flits coming from R_{q-1} enter router R_q via port W and conflict with the injection of flits by π_q in any time interval of length t . We derive an upper-bound on $\hat{n}_{q^w}(t)$ below; for now, let's assume its value is known. Consequently, π_q can only attempt to inject flits during the remaining $t - \hat{n}_{q^w}(t)$ clock cycles.

However, according to our strategy to regulate traffic injection, π_q suffers the longest delay in injecting a flit of $f_{q,i}$ when flits coming from π_{q-1} prevent π_q from injecting flits, while the PE directly east π_{q+1} asks for free-slots to inject its own flits. In this scenario, π_q will use only some of the free-clock cycles it receives from π_{q-1} and pass the rest to π_{q+1} . To fulfill this request, π_q specifically allocates D_q out of every G_q free clock cycles it receives from π_{q-1} , passing the remainder to π_{q+1} . In the worst-case, at the start of the $t - \hat{n}_{q^w}(t)$ clock cycles, π_q has already utilized D_q credits to inject its own flits, leaving the remaining $G_q - D_q$ clock cycles for π_{q+1} to use. Therefore, during the first $G_q - D_q$ initial clock cycles of the $t - \hat{n}_{q^w}(t)$ under analysis, π_q cannot inject flits. π_q can now only attempt to inject flits in the remaining $t - \hat{n}_{q^w}(t) - (G_q - D_q)$ clock cycles.

D_q new credits are generated after those $G_q - D_q$ cycles. In $t - \hat{n}_{q^w}(t) - (G_q - D_q)$ clock cycles, we can identify $\left\lfloor \frac{t - \hat{n}_{q^w}(t) - (G_q - D_q)}{G_q} \right\rfloor$ intervals of G_q clock cycles. In each of these intervals, π_q injects D_q flits, which is equivalent to $D_q \cdot \left\lfloor \frac{t - \hat{n}_{q^w}(t) - (G_q - D_q)}{G_q} \right\rfloor$ flits in total. Hence, π_q injects $D_q \cdot \left\lfloor \frac{t - \hat{n}_{q^w}(t) - (G_q - D_q)}{G_q} \right\rfloor + \min(D_q, (t - \hat{n}_{q^w}(t) - (G_q - D_q)) \bmod G_q)$ flits in t in the worst-case. The term $\min(D_q, (t - \hat{n}_{q^w}(t) - (G_q - D_q)) \bmod G_q)$ represents the remaining clock cycles after calculating how many intervals of size G_q are in $t - \hat{n}_{q^w}(t) - (G_q - D_q)$ clock cycles.

Therefore, in the worst-case scenario, π_q requires t clock cycles to inject $C_{q,i}$ flits into the network, where t satisfies the following equation:

$$C_{q,i} = D_q \cdot \left\lfloor \frac{t - \hat{n}_{q^w}(t) - (G_q - D_q)}{G_q} \right\rfloor + \min(D_q, (t - \hat{n}_{q^w}(t) - (G_q - D_q)) \bmod G_q) \quad (6.2)$$

Hence, the lemma is proven. $\square$

Let's now derive an upper-bound on $\hat{n}_{q^w}(t)$. In a time interval of length t , the maximum number of flits $\hat{n}_{q^w}(t)$ that can arrive at π_q via port W and prevent flit's injection is given by the sum of: (1) the maximum number of flits $\hat{n}_{q-1}(t)$ injected by π_{q-1} in that interval when flits coming from the PE directly west π_{q-2} prevent π_{q-1} from injecting flits and the PE directly east π_q asks for free-slots to inject its own flits, and (2) the maximum number of flits $\hat{n}_{q-1^w}(t)$ that arrive at π_{q-1} via port W . That is,

$$\hat{n}_{q^w}(t) = \hat{n}_{q-1}(t) + \hat{n}_{q-1^w}(t) \quad (6.3)$$

Lemma 26 derives an upper-bounds on $\hat{n}_{q-1}(t)$.

Lemma 26. In any time interval of length t , the maximum number of flits a PE π_{q-1} can inject when flits from π_{q-2} conflict with their injections and π_q requests free-slots, is defined by:

$$\hat{n}_{q-1}(t) = D_{q-1} \cdot \left\lfloor \frac{t - \hat{n}_{q-1^w}(t)}{G_{q-1}} \right\rfloor + \min(D_{q-1}, (t - \hat{n}_{q-1^w}(t)) \bmod G_{q-1}) \quad (6.4)$$

Proof. As discussed in the previous Lemma, within a time interval of length t , π_{q-1} can attempt to inject flits in $t - \hat{n}_{q-1^w}(t)$ clock cycles. Then, π_{q-1} injects the maximum number of flits if at the beginning of the $t - \hat{n}_{q-1^w}(t)$ clock cycles, π_{q-1} has used no credit to inject its own flits. This results in $\left\lfloor \frac{t - \hat{n}_{q-1^w}(t)}{G_{q-1}} \right\rfloor$ intervals of G_{q-1} clock cycles, where, as π_{q-1} can only inject D_{q-1} flits every G_{q-1} clock cycles, a total of $D_{q-1} \cdot \left\lfloor \frac{t - \hat{n}_{q-1^w}(t)}{G_{q-1}} \right\rfloor + \min(D_{q-1}, (t - \hat{n}_{q-1^w}(t)) \bmod G_{q-1})$ flits are injected in t . The term $\min(D_{q-1}, (t - \hat{n}_{q-1^w}(t)) \bmod G_{q-1})$ accounts for the remaining clock cycles after determining the number of G_{q-1} -sized intervals within $t - \hat{n}_{q-1^w}(t)$ clock cycles. Thus, the lemma is proven. $\square$

Note that, $\hat{n}_{q^w}(t)$ is dependent on $\hat{n}_{q-1^w}(t)$ which can directly or transiently be dependent on $\hat{n}_{q^w}(t)$. Therefore, there is a cyclic dependency, and finding a bound on $\hat{n}_{q^w}(t)$ requires to solve a set of equations using a fixed point iteration. Note that, as $\hat{n}_{q-1^w}(t)$ decreases, the function also decreases, and is lower bounded by 0. This means that the function will eventually reach a stable point.

The proposed analysis makes the assumption that PEs inject flits in every free-clock cycle they reserve for themselves. However, this assumption does not always hold true. For instance, there might be situations where certain PEs do not release any packets during the analyzed time interval, which means they will not be injecting flits. While this analysis may seem pessimistic under certain conditions, it can be highly valuable in cases where comprehensive system information is lacking. The analysis relies solely on information related to the communication flow being analyzed and the PEs connected to the network. This quality makes the approach agnostic and well-suited for dynamic applications where the set of communication flows can change over time.

6.5 Experimental Results

In this section, we evaluate the behavior of the proposed traffic injection regulation protocol. In particular, we analyze the number of flits that PEs connected to a HopliteRT* NoC inject in a given time interval when our solution is integrated into the design.

6.5.1 Synthetic Test Cases

We consider different scenarios to analyze the impact of our solution on PEs depending on their position in the network. We first consider a 4×4 HopliteRT* NoC and three communication flows injected by three different PEs with origin at coordinates (0;2), (1;2), and (2;2), respectively (i.e., the origin PEs are on the same row of the grid-based network), and destination at coordinate (3;0) for the three flows (see Figure 6.5a). We simulate a case where every PE tries to inject flits as fast as possible, thus trying to use as much bandwidth as available. The flit size was set to 64 bits.

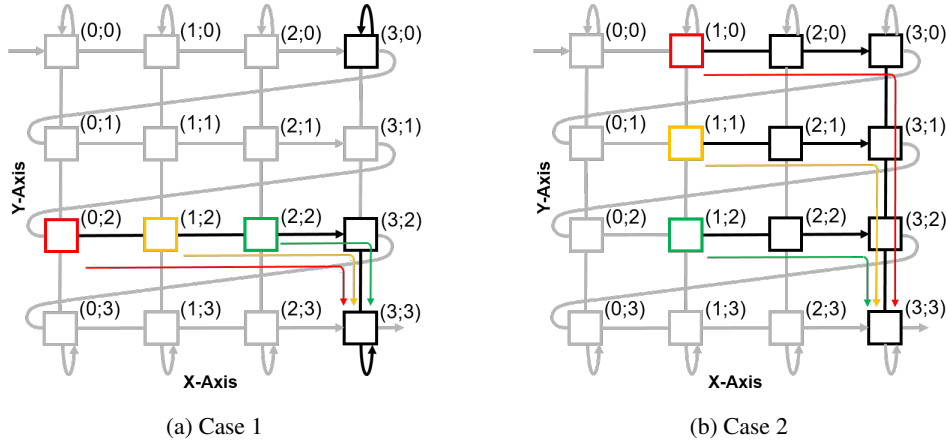


Figure 6.5: Test cases.

Figure 6.6 shows the number of flits PEs injected in given time intervals. In HopliteRT*, as we observe in Figure 6.6a, the flow injected by PE at coordinate (0;2) sends flits significantly faster against the other flows. This can be explained by the fact that it does not conflict with any other flow traversing the network when accessing its own output port. Therefore, it injects a flit every clock cycle. However, this is not the case for the flows injected by the PEs at coordinates (1;2) and (2;2). Specifically, the flow released at (1;2) is prevented from injecting flits until PE at (0;2) has injected all its flits, since the flow coming from PE at (0;2) accesses network resources first and therefore has the highest priority for using port *E* at (1;2) once traversing the network. The same happens to the flow injected by PE at (2;2), which must wait until the other two PEs have injected all their flits to start injecting its own flits.

However, this is not the case when our traffic injection regulation protocol is added to HopliteRT*. As shown in Figure 6.6b, when the three PEs inject flits on 2 out of 10 free-clock cycles, PE at (2;2) injects flits faster. This follows from the fact that PEs at (0;2) and (1;2) constantly receive requests from the PEs directly to the east to free bandwidth, and therefore, their injection

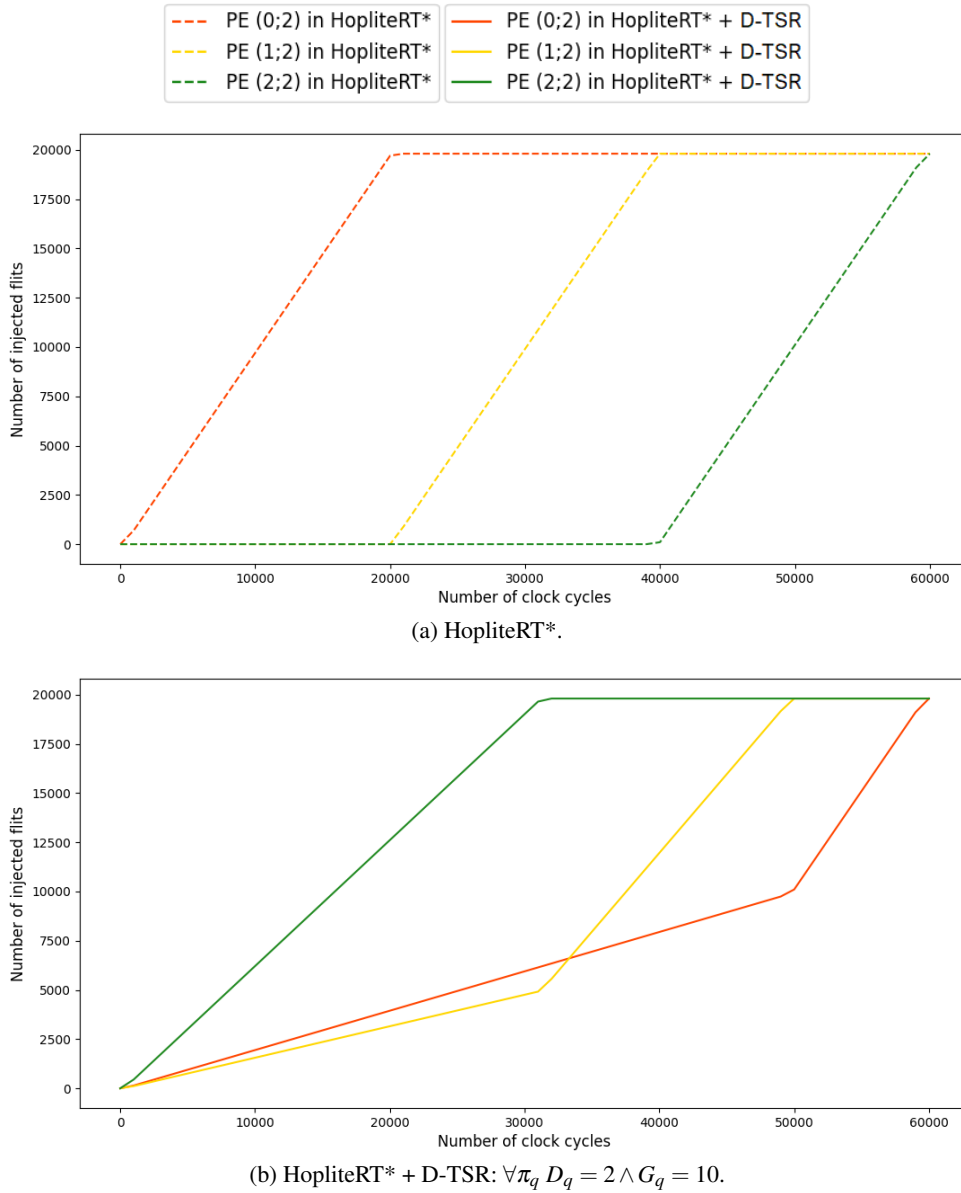


Figure 6.6: Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 100.

rate is controlled and limited to only 2 out of 10 free-slots. Conversely, PE at (2;2) uses every free-clock cycle it gets, as no packet releases in the PE to its east. Note that, once PE at (2;2) has injected all packets, PE at (1;2) injects flits in a faster fashion. This is because PE at (1;2) no longer needs to pass free-clock cycles to PE at (2;2) and can therefore use any free-slot it receives from PE at (0;2) for itself. In Figure 6.6c, we show the results obtained when the three PEs are configured to use 5 out of 10 free-clock cycles. Note that, PE at (1;2) and PE at (2;2) inject flits at the same rate. In Figure 6.6d, we then provide the result for the scenario where the PEs use 9 out of 10 free-clock cycles. As we observe, in Figure 6.6, the results obtained when adding our approach to HopliteRT* converge to those of HopliteRT* when the number of free-slots PEs can use

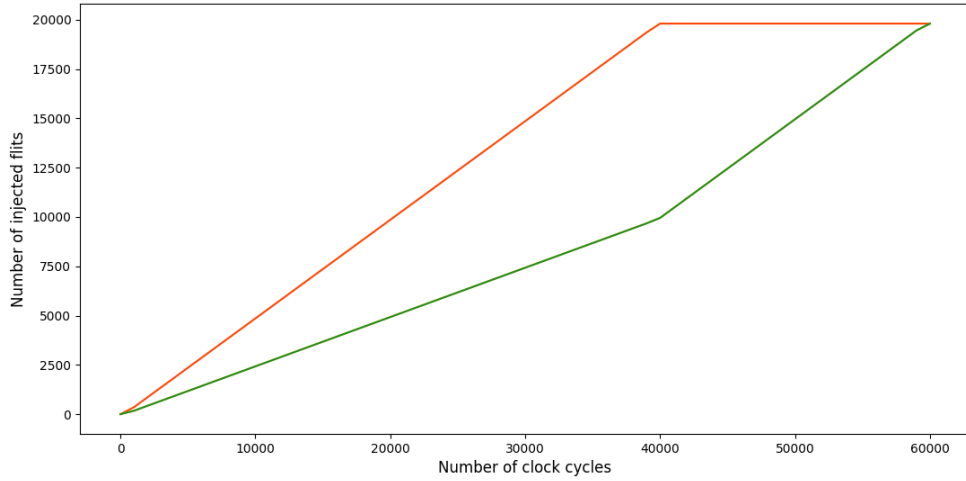
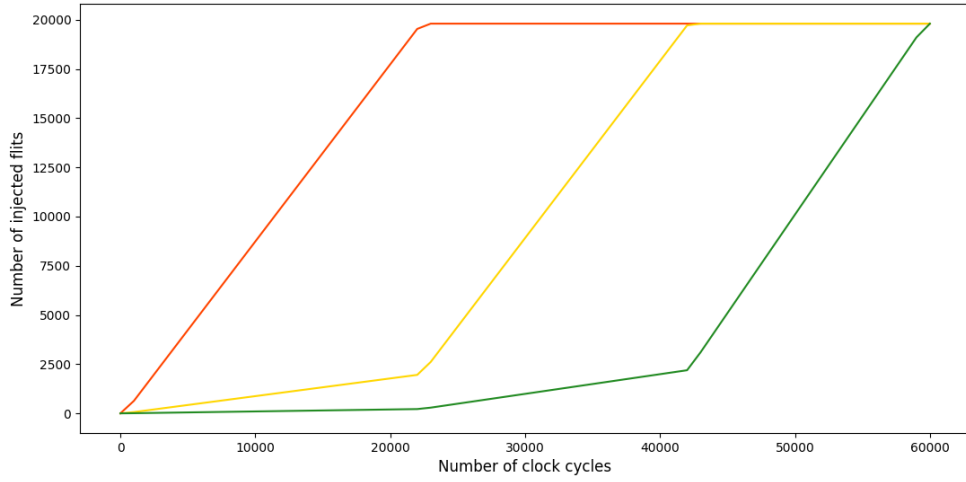
(c) HopliteRT* + D-TSR: $\forall \pi_q D_q = 5 \wedge G_q = 10$.(d) HopliteRT* + D-TSR: $\forall \pi_q D_q = 9 \wedge G_q = 10$.

Figure 6.6: Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 100 (cont.).

is increased. In conclusion, while HopliteRT* favors PEs located upstream, thus generating long, potentially unbounded blocking time for routers downstream (as in Figure 6.6a), traffic regulation protocol, allows a more equitable and controllable distribution of network bandwidth between the PEs. Indeed, the three communication flows can send flits through the network concurrently, with a rate configurable using the parameters D and G in each router.

In Figure 6.7, we then show the results obtained when the utilization of the three PEs was set 20% (i.e., they release 60 flits every 300 clock cycles). As we observe, both solutions yield practically the same values, even when PEs' injection rate changes. Indeed, the three communication flows consume the same number of clock cycles when injecting flits into the network. This can be explained by the fact that the level of congestion over shared resources decreases as flows need to inject fewer flits in each time interval. In this case, PEs are not forced to request many free-time slots to inject their flits, and hence, flits do not suffer long waits inside PEs before being injected.

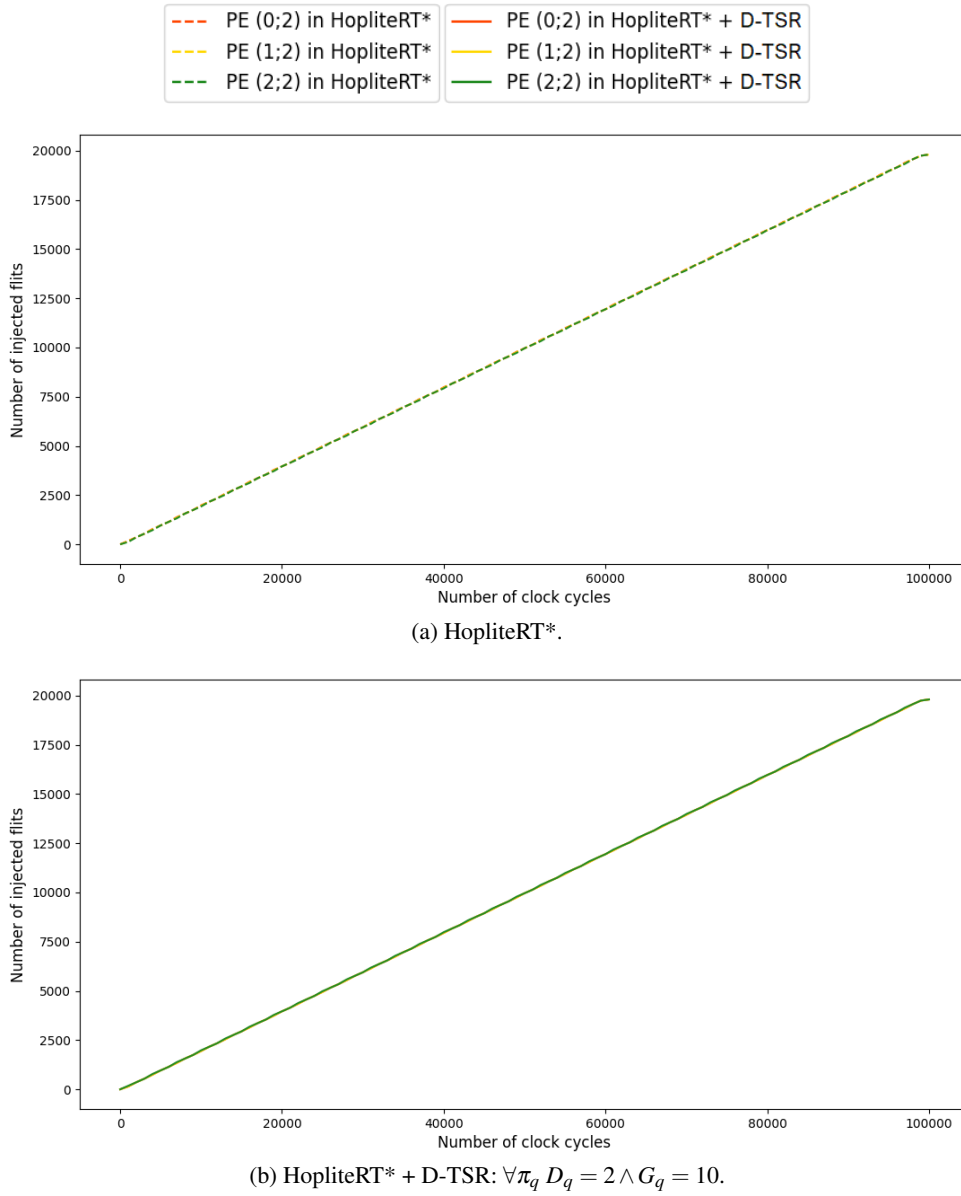


Figure 6.7: Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 20.

In a second set of experiments, we consider three communication flows with origin PEs at coordinates (1;0), (1;1), and (1;2), respectively (i.e., the origin PEs are on the same column of the grid-based network instead of the same row), and destination at coordinate (3;3) for the three flows (see Figure 6.5b). PEs inject flits as fast as possible. The rest of the parameters remain the same as in the previous scenario.

In HopliteRT*, flows coming from PE at (1;0) and PE at (1;1) conflict only when they are on their way south to their destination router. Thus, both PE at (1;0) and PE at (1;1) are able to inject a new flit to port *E* of its associated router in each clock cycle (see Figure 6.8a). However, this is not the case for PE at (1;2). Flits coming from PE at (1;0) and PE at (1;1) conflict for port *S* of

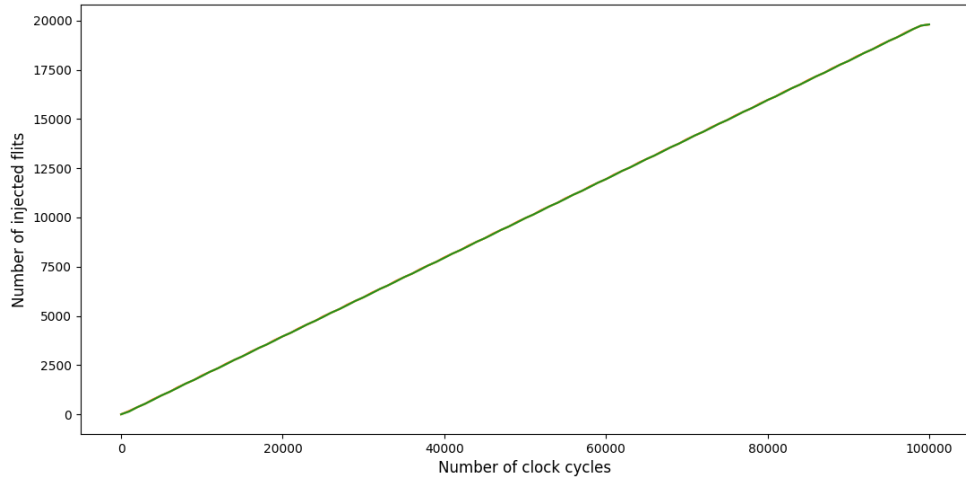
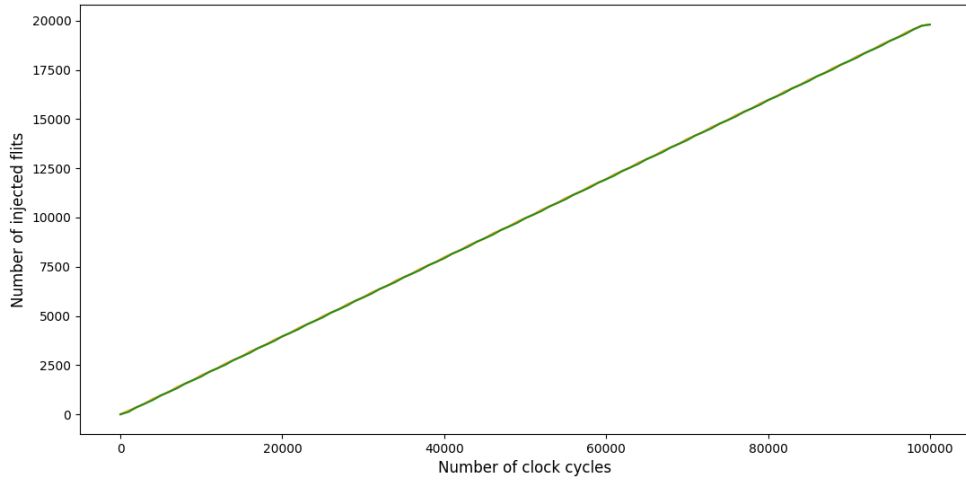
(c) HopliteRT* + D-TSR: $\forall \pi_q D_q = 5 \wedge G_q = 10$.(d) HopliteRT* + D-TSR: $\forall \pi_q D_q = 9 \wedge G_q = 10$.

Figure 6.7: Number of injected flits in given time intervals when PEs are on the same row of the network and have utilization equal to 20 (cont.).

router at (3;1). In this contention scenario, one of the flits is routed through the desired output port while the other is deflected to the east. The deflected flits then travel along the X-axis until they reach the same router they would have if they had not been deflected. That is, they enter the router at (1;2) via port *W* and request port *E*, which prevents PE from injecting flits until the requested port becomes free, thus increasing the number of clock cycles flits must wait within PE before being injected. This happens as soon as PE at (1;0) and PE at (1;1) have no more flits to inject. Under our solution, however, PE at (1;2) requests free-time slots to inject flits to the PE directly to the west. Such requests are then forwarded westward until they arrive at PE at (1;1). Then, PE at (1;1) will use only a portion of the total number of free-slots it gets to inject its own flits and pass the remainder to meet the requests of PE at (1;2). Note that, if PE at (1;1) injects no flit, there is no contention scenario over port *S* of the router at (3;1), and therefore, there is no deflected flit blocking flit injection by PE at (1;2). Note also that, PE at (1;0) does not see modified its injection

rate, i.e., it continues to inject one flit per clock cycle (see Figures 6.8b, 6.8c, and 6.8d).

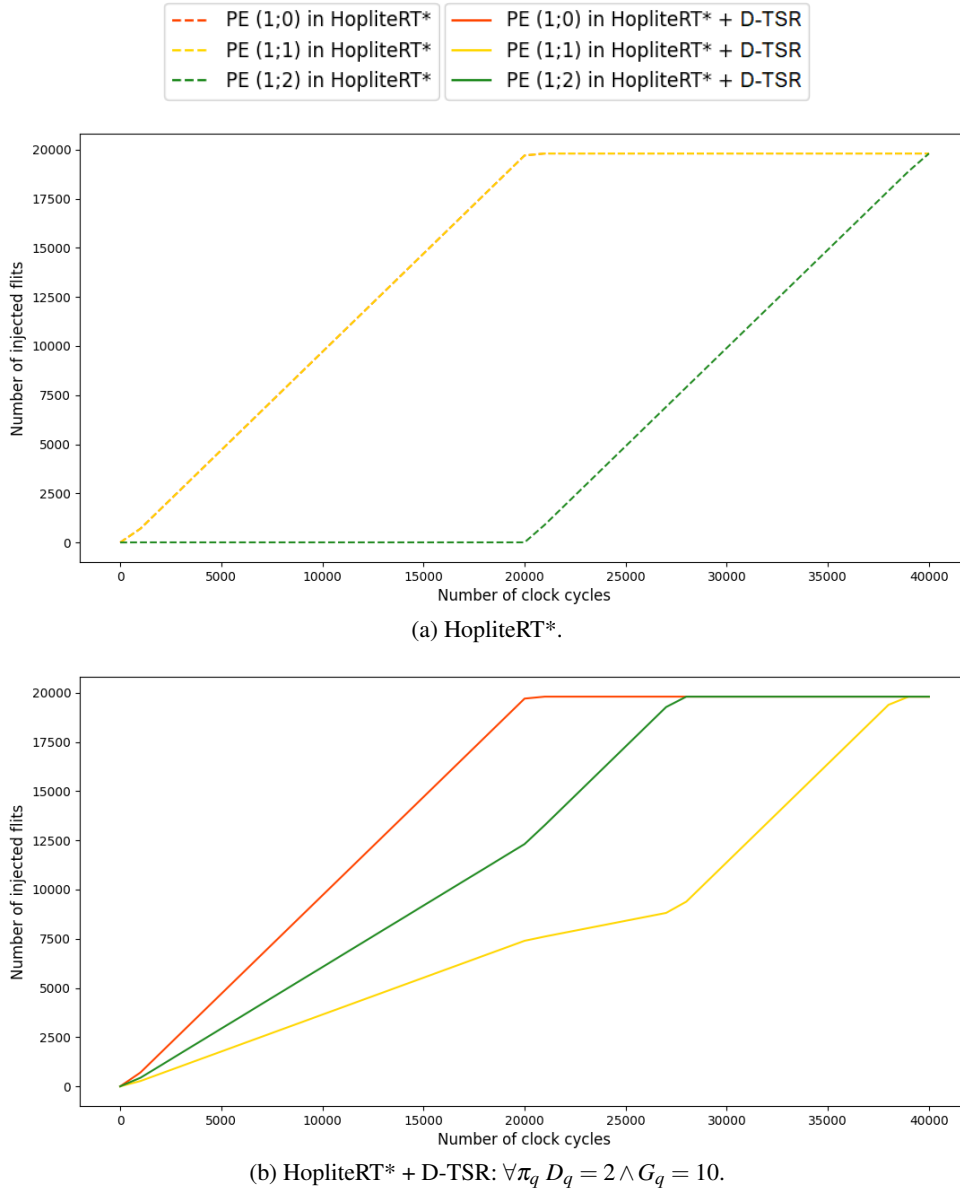


Figure 6.8: Number of injected flits in given time intervals when PEs are on the same column of the network and have utilization equal to 100.

In a third set of experiments, we consider the all-to-one traffic pattern where all PEs of a 4×4 NoC inject flits as fast as possible towards the same destination at coordinate (3;3). Figure 6.9 shows the number of flits each PE injects in given time intervals under HopliteRT* and HopliteRT* + D-TSR. As we observe in Figure 6.9a, in HopliteRT*, the injection of flits by PEs follows a specific pattern. PEs placed upstream in areas of the network with low congestion start injecting their flits first and prevent other PEs from injecting flits until all their transmissions are completed. For instance, the PE at coordinate (1;1) initiates flit injection only after the PE at coordinate (0;1) has injected all its flits. However, when our proposed solution is integrated into HopliteRT*,

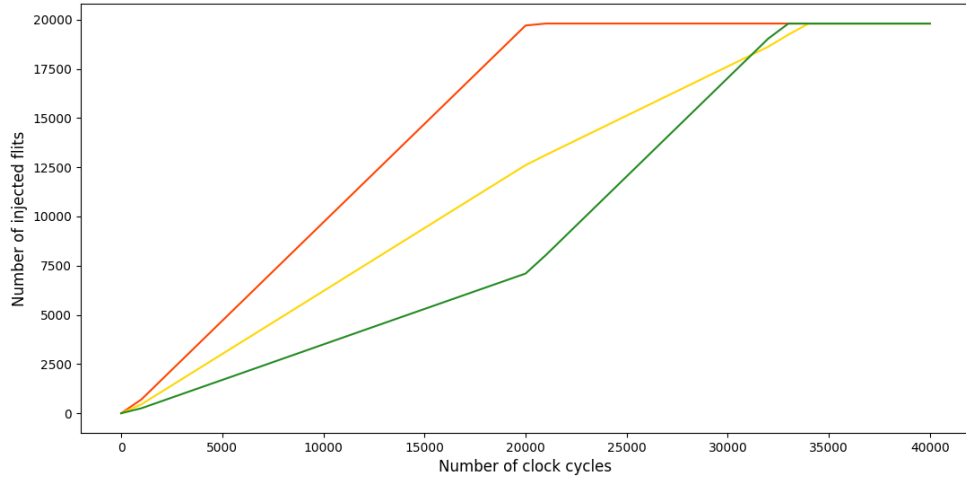
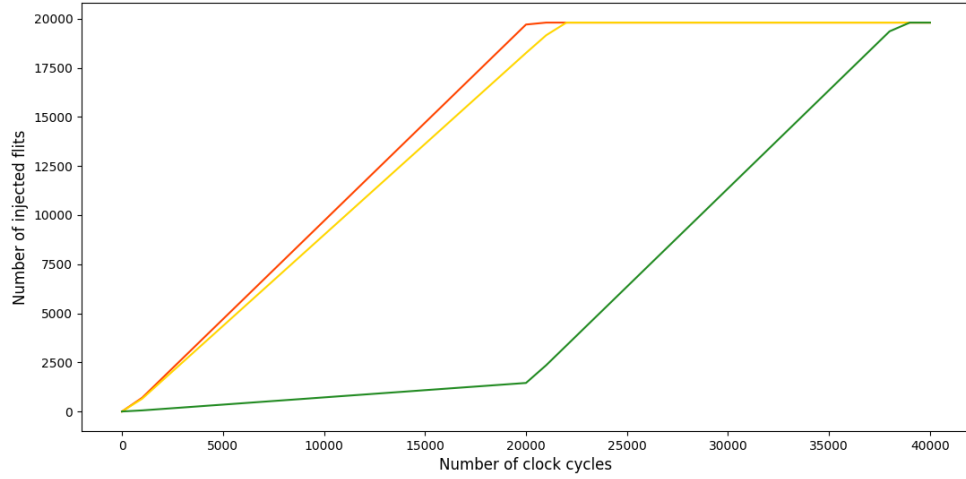
(c) HopliteRT* + D-TSR: $\forall \pi_q D_q = 5 \wedge G_q = 10$.(d) HopliteRT* + D-TSR: $\forall \pi_q D_q = 9 \wedge G_q = 10$.

Figure 6.8: Number of injected flits in given time intervals when PEs are on the same column of the network and have utilization equal to 100 (cont.).

a notable change occurs. In this case, the injection of flits by PEs is no longer determined by their position in the network. Instead, the new solution achieves a more equitable and balanced utilization of the available network bandwidth by dynamically adjusting PEs' injection rates (see Figures 6.9b- 6.9d).

6.5.2 Orion Test Case

We use the Orion Crew Exploration Vehicle case study [71] to evaluate our solution.

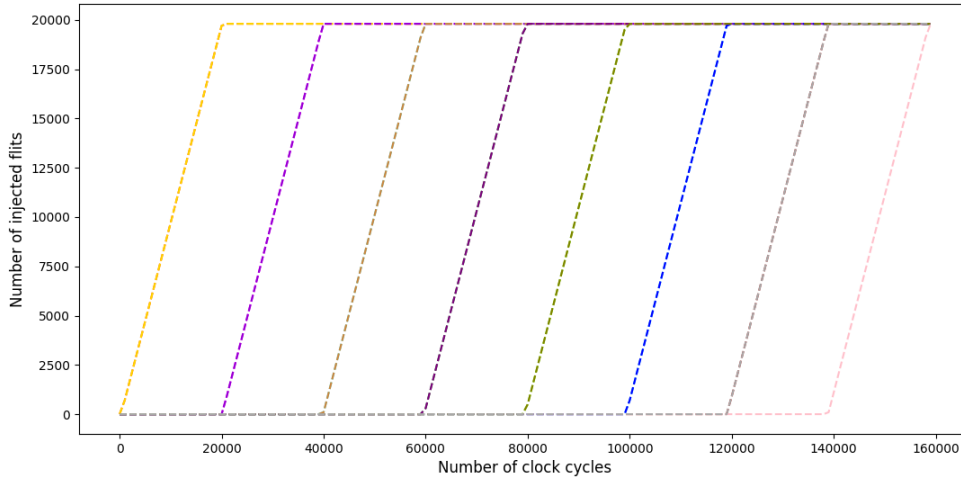
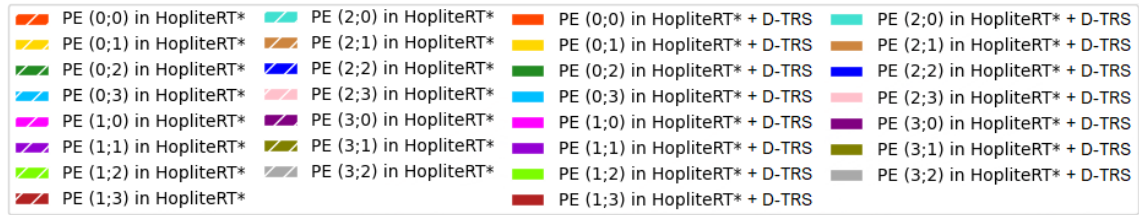
Table 6.1 shows the configuration details for the Orion test case. The parameters of the traffic flows were defined considering those in [85]. We configured each PE to generate between 2 and 10 communication flows into the network. The communication flows were generated according to a random traffic pattern. That is, the source and destination PEs of each flow were randomly generated. The utilization U_q of each PE π_q was set to 100%, while G_q was set 10 and D_q was

randomly chosen within the set $\{2; 3; \dots; 9\}$. The utilization $U_{q,i}$ of each flow $f_{q,i}$ was then generated using a uniform probability distribution, where the sum of the utilization of all flows mapped in a PE π_q is U_q . The inter-arrival time $T_{q,i}$ of each flow $f_{q,i}$ was randomly chosen within the interval $[400, 37500]$ (see Table 6.1). The number of flits $C_{q,i}$ per packet was then determined by multiplying $U_{q,i}$ by $T_{q,i}$. The flit size was set at 64 bits.

Table 6.1: Configuration details for Orion test case.

Network size	Number of PEs	Number of PEs generating flows	Number of flows	T_{min} (Clock cycles)	T_{max} (Clock cycles)
6x6	36	31	187	400	37500

Figure 6.10 shows the results for the Orion test case when communication flows pass through HopliteRT* and HopliRT* + D-TSR NoCs. As can be seen in Figure 6.10a, multiple PEs can only start injecting flits when other PEs have no more flits to inject when D-TSR is not used. However, this is not the case when HopliteRT* supports our traffic injection regulation solution. That is, the amount of network bandwidth that each PE can use in each time interval is controlled by the G_q and D_q parameters.



(a) HopliteRT*.

Figure 6.9: Number of injected flits in given time intervals under all-to-one traffic pattern when all PEs have utilization equal to 100.

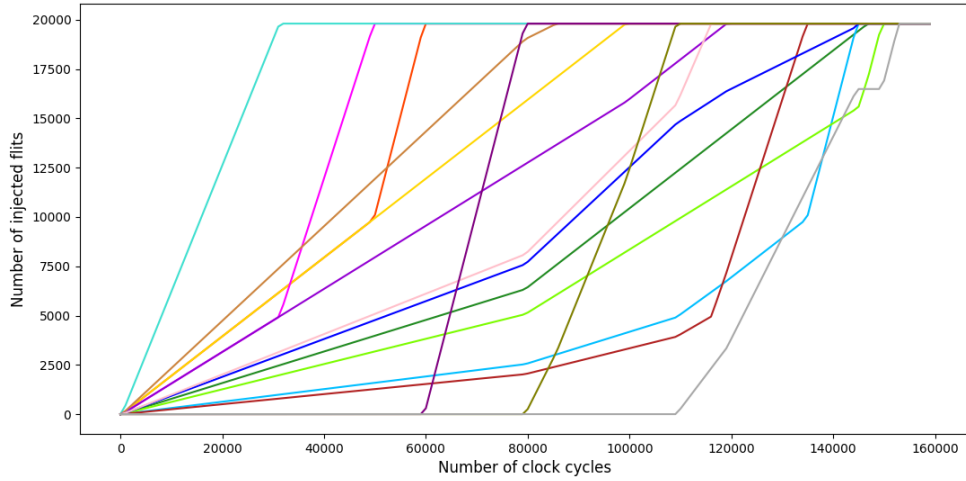
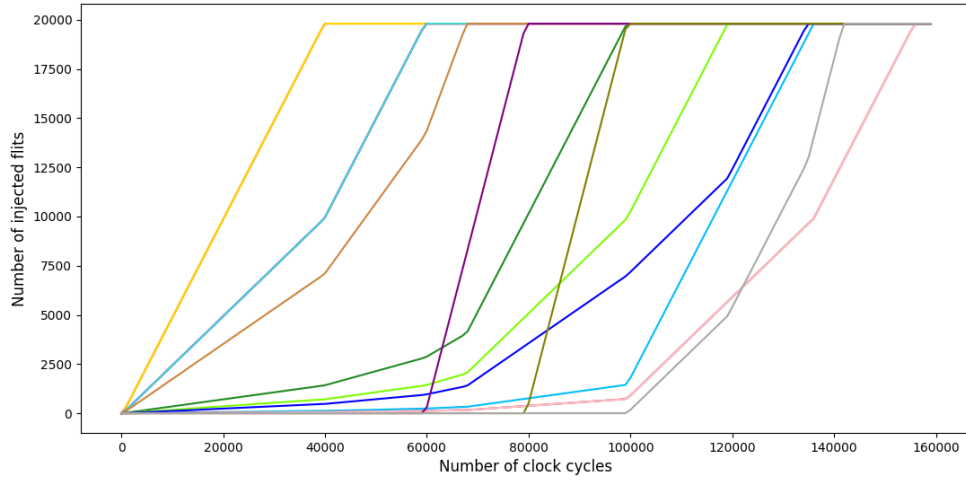
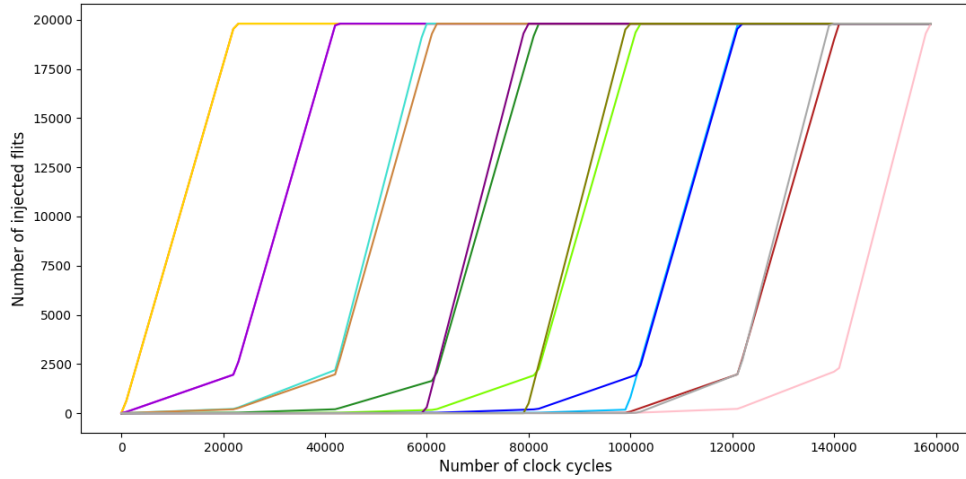
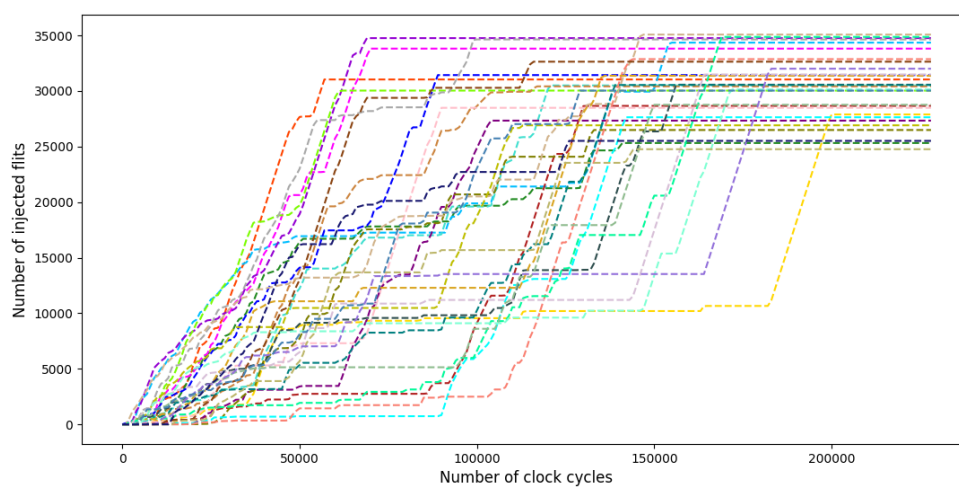
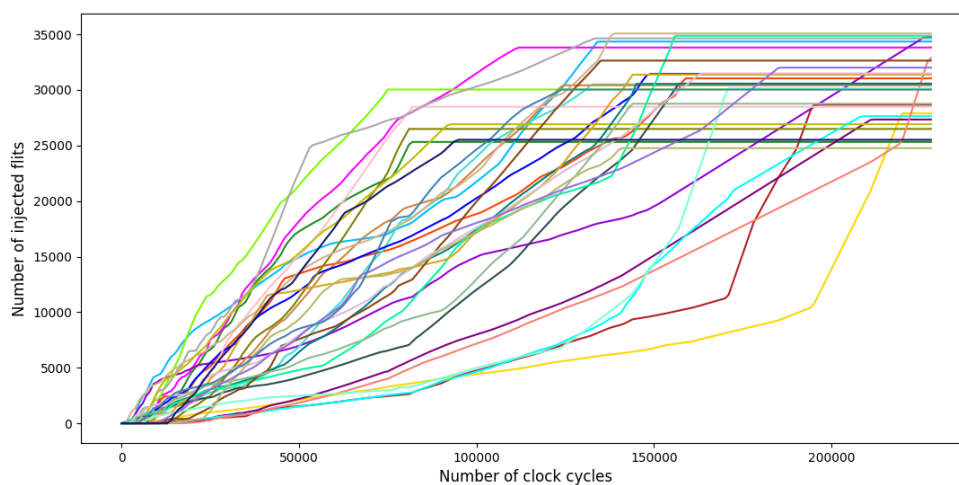
(b) HopliteRT* + D-TSR: $\forall \pi_q D_q = 2 \wedge G_q = 10$.(c) HopliteRT* + D-TSR: $\forall \pi_q D_q = 5 \wedge G_q = 10$.(d) HopliteRT* + D-TSR: $\forall \pi_q D_q = 9 \wedge G_q = 10$.

Figure 6.9: Number of injected flits in given time intervals under all-to-one traffic pattern when all PEs have utilization equal to 100 (cont.).



(a) HopliteRT*.

Figure 6.10: Number of injected flits for the Orion test case.



(b) HopliteRT* + D-TSR.

Figure 6.10: Number of injected flits for the Orion test case (cont.).

6.6 Chapter Summary

In this chapter, we presented a novel strategy, based on dynamic time slot requests, for regulating traffic injection as a function of network congestion. The approach tackles the problem of certain PEs being blocked indefinitely due to unfair bandwidth usage by other PEs. By delaying the injection of certain flits into the network, we achieved a more balanced distribution of network bandwidth among all PEs. This leads to a more effective utilization of network links, resulting in lower traffic delays and enhancing the overall performance of the system. Moreover, since the proposed solution uses run-time information, it does not waste communication bandwidth. We also derived a new upper-bound on the WCIT of flits when traffic injection is regulated by the D-TSR protocol. Two sets of experiments demonstrated that our solution achieves a more equitable distribution of network bandwidth compared to mechanisms based on the leaky bucket approach.

Chapter 7

Attempt to Generalize to a Higher-Dimensional NoC Architecture

The continuous increase in the number of processing elements and communication flows in modern MPSoCs, driven by the escalating complexity of applications, presents a significant challenge for traditional network topologies such as mesh or torus. Recent studies, as demonstrated in [76], indicate that circulant topologies exhibit superior characteristics compared to those of more traditional topologies in non-real-time scenarios. Hence, by incorporating these higher-dimensional network topologies into NoC designs, we create new opportunities to enhance the efficiency and scalability of MPSoCs.

The potential benefits of higher-dimensional network topologies are considerable. One of their main advantages is the improved connectivity they offer. By adding extra dimensions, routers can be connected to more routers in more directions, providing multiple paths for flits to traverse the network. As a result, flits can take advantage of the numerous available routes, potentially reducing the number of hops needed to traverse the network from their origin to their destination. Moreover, the increased number of links and routing options enables greater adaptability to changing traffic patterns and communication demands.

However, despite the advantages offered by higher-dimensional NoCs, they also come with challenges. The increased complexity of these network topologies may lead to the development of more complex and sophisticated routing algorithms, as well as the requirement for additional hardware resources. Designing and implementing NoCs based on such topologies requires careful consideration of the trade-offs between performance gains and resource overhead.

This chapter presents nDimNoC, a flexible higher-dimensional NoC architecture (where the number of dimensions can be greater than or equal to 2) that provides real-time guarantees for systems implemented upon MPSoCs. It aims to reduce average network communication latency and offers greater flexibility compared to more traditional 2D-NoCs. Specifically, we extend the deflection-based routing policy introduced in HopliteRT* NoC, taking advantage of the favorable properties of circulant topologies to ensure bounded worst-case communication delays.

The chapter is structured as follows. Section 7.1 provides an overview of related work available in the literature, and the system model is presented in Section 7.2. In Section 7.3, we introduce the nDimNoC architecture, and in Section 7.4, we derive a timing analysis based on DAG for packets transmitted over nDimNoC. Finally, in Section 7.5, we present and discuss several experimental results. We assess our new design against related works in terms of computed WCCT bounds and hardware requirements.

The content of this chapter was previously published in the following paper:

- **Y. Ribot González**, G. Nelissen, and E. Tovar. nDimNoC: Real-Time D-dimensional NoC. In 33rd Euromicro Conference on Real-Time Systems (ECRTS'21), vol. 196, pp. 5:1-5:22, 2021, doi:10.4230/LIPIcs.ECRTS.2021.5

7.1 Related Work

From a 3D-NoC perspective, Park et al. [68] proposed a Multi-layered on-chip Interconnect Router Architecture (MIRA). Their approach assumes 3D processor designs (i.e., processor cores partitioned into multiple layers), and is therefore inadequate for existing highly optimized 2D processor designs. In [29], Ghidini et al. presented a 3D-NoC mesh architecture called Lasio relying on wormhole switching with FIFO queues. In order to minimize packet communication latency and NoC area, Tiny 3D mesh NoC was later proposed in [57]. Tiny NoC reduces the number of routers and links in the network by connecting multiple programming elements to the same router. This solution minimizes the total NoC area as compared to Lasio NoC, however, average packet latency improves only when there are few flows and/or under a low packet injection rate. In [15], a 3D NoC architecture based on De-Bruijn graph was proposed. Tree-based interconnect architectures have been also considered in some works [36, 37, 35]. However, they are very complex to implement due to their irregular and complex network topologies. In [26], a NoC/Bus-based hybrid 3D architecture was proposed, but the approach suffers from low throughput due to inefficient hybridization between the NoC and bus media.

7.2 System Model

We assume a system composed of N programming elements $\{\pi_1, \dots, \pi_N\}$. Each programming element π_q is connected to a different router R_q of a higher-dimensional NoC. In a network with D dimensions (i.e., a D -dimensional network), the coordinates of a router R_q are noted $(r_1^q, r_2^q, \dots, r_D^q)$. Each programming element π_q injects a set of n^q communication flows $F^q = \{f_1^q, f_2^q, \dots, f_{n^q}^q\}$ into the network. A communication flow f_j is defined by the parameters $\{(s_1^j, s_2^j, \dots, s_D^j), (d_1^j, d_2^j, \dots, d_D^j), C_j, T_j\}$. A communication flow f_j generates a potentially infinite number of packets that are injected at coordinates $(s_1^j, s_2^j, \dots, s_D^j)$ of the NoC and must reach the programming element at coordinates $(d_1^j, d_2^j, \dots, d_D^j)$. f_j respects a minimum inter-arrival time T_j between the generation of every two packets. Each packet sent by flow f_j is divided in C_j flits that are sequentially injected in the network. Each flit has a size S_{flit} (in bits). We assume that all the routing information is encoded

in each flit of the packet, i.e., there is no distinction between header, body or tail flits. The routing information is the coordinates of the destination programming element of the associated flow.

In the rest of the chapter, we use the notations $R_{orig}(f_j)$ and $R_{dest}(f_j)$ to refer to the origin and destination router of flow f_j , respectively. That is, $R_{orig}(f_j)$ has coordinates $(s_1^j, s_2^j, \dots, s_D^j)$ and $R_{dest}(f_j)$ has coordinates $(d_1^j, d_2^j, \dots, d_D^j)$.

7.3 NoC Design

7.3.1 Network Topology

Consider a network composed of N routers R_0 to R_{N-1} . In nDimNoC, the routers are connected together according to a ring circulant topology $C(N; g_1, g_2, \dots, g_D)$ where $g_1 = 1$, N is the total number of routers, D indicates the dimensionality of the network, and $g_1, g_2, \dots, g_D$ are the *generatrices* of the network. We assume that the generatrices follow the following properties: $1 = g_1 < g_2 < \dots < g_D < N$, and that their values are harmonic (i.e., for any pair of generatrices g_i and g_j such that $i < j$, g_i is a divider of g_j). Under the circulant topology $C(N; 1, g_2, \dots, g_D)$, all routers have D inputs $I_1, I_2, \dots, I_D$ and D outputs $O_1, O_2, \dots, O_D$ for inter-routers communications. All routers are connected by a single unidirectional ring using one of their inputs and one of their outputs (see blue line in Figure 7.1a). Then, each router is also connected to the routers that are $g_2, g_3, \dots, g_D$ hops away on the ring (see red, green and black lines in Figure 7.1a). Formally stated, for each router R_q (with $0 \leq q < N$), its u^{th} output port O_u ($1 \leq u \leq D$) is connected to the u^{th} input port I_u of the router $R_{(q+g_u) \bmod N}$.

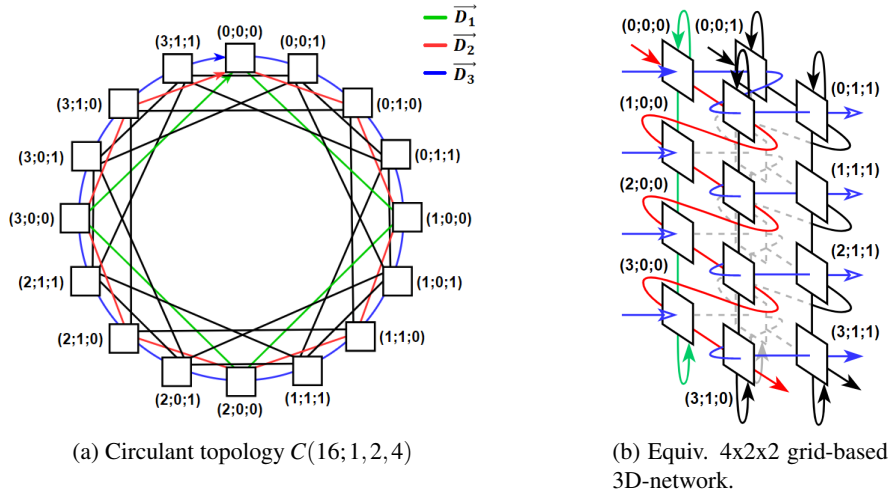


Figure 7.1: nDimNoc network topology.

A circulant network $C(N; 1, g_2, \dots, g_D)$ may also be represented as a $S_1 \times S_2 \times \dots \times S_D$ grid-based D -dimensional network, where $S_1, S_2, \dots, S_D$ correspond to the number of routers on the dimension $\vec{D_1}, \vec{D_2}, \dots, \vec{D_D}$, respectively. The size of the network on each dimension can be computed as follows

$S_1 = \frac{N}{g_D}, S_2 = \frac{g_D}{g_{D-1}}, S_3 = \frac{g_{D-1}}{g_{D-2}}, \dots, S_D = \frac{g_2}{g_1}$. The coordinates $(r_1^q, r_2^q, \dots, r_D^q)$ of a router R_q defines the position of the router R_q in the grid representation.

As an example, Figure 7.1a shows the circulant network $C(16; 1, 2, 4)$. In Figure 7.1b, we provide the equivalent representation as a 4x2x2 grid-based 3-dimensional network of the circulant network shown in Figure 7.1a. The red, green, and blue links in Figure 7.1a correspond to the red, green, and blue links in Figure 7.1b, respectively.

In the rest of this paper, we often reason about the position pos^q of a router R_q on the main unidirectional ring of the circulant topology. That position can be inferred from the coordinates of the router in the grid topology as follows

$$pos^q = \sum_{k=1}^D r_k^q \times g_{D-k+1} \quad (7.1)$$

To simplify some of our further discussions, we define the helping function $\text{dist}(R_q, R_m)$ as the distance between routers R_q and R_m on the main ring, i.e.,

$$\text{dist}(R_q, R_m) = (pos^m - pos^q + N) \bmod N \quad (7.2)$$

Note that the following properties hold for circulant topologies.

Property 5. Let R_l be a router at which flit p is located. After one hop on dimension $\vec{D}_u$ of the network, flit p reaches a router R_m located g_{D-u+1} steps further on the main ring of the network, i.e., $\text{dist}(R_l, R_m) = g_{D-u+1}$.

Finally, we define $\text{ring}_u(R_q)$ as the set of routers that are on the same ring of dimension $\vec{D}_u$ as R_q . That is,

$$\text{ring}_u(R_q) = \{R_l \mid \forall b \in [u+1, D], r_b^l = r_b^q\} \quad (7.3)$$

As an example, let R_0 be the router at coordinates $(0; 0; 0)$ in Figure 7.1a, then all the routers connected by the green links are in $\text{ring}_1(R_0)$, and all the routers connected by red links are in $\text{ring}_2(R_0)$.

7.3.2 Router Architecture

nDimNoC generalizes the architecture and routing policy introduced in HopliteRT*. As we discussed in the previous section, nDimNoC routers have D inputs (i.e., $I_1, I_2, \dots, I_D$) and D output ports (i.e., $O_1, O_2, \dots, O_D$) connected to neighboring routers to allow for inter-routers communication (see Figure 7.2). In addition, all routers also have D input ports (i.e., $I_1^{PE}, I_2^{PE}, \dots, I_D^{PE}$) that may be used by the programming element to inject packets into the network. Therefore, in total, each router has $2 \times D$ input ports and D output ports. A programming element can inject packets on any of the D dimensions $\vec{D}_1, \vec{D}_2, \dots, \vec{D}_D$ of the network by using the input ports $I_1^{PE}, I_2^{PE}, \dots, I_D^{PE}$, respectively. Therefore, several packets may be injected into different dimensions simultaneously. Thus, the waiting times suffered by the packets inside the programming elements decrease. Indeed, in solutions that support a single input port to inject packets into the network, all packets

compete for the same input port. In nDimNoc, however, a packet that is waiting to be injected into the network only conflicts with the subset of packets that must be injected to the same input port I_u^{PE} .

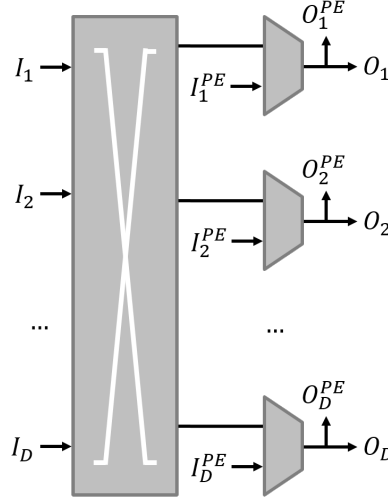


Figure 7.2: nDimNoc router architecture.

Property 6. In this paper, we assume that a flit of a flow f_j with origin and destination coordinates $(s_1^j, s_2^j, \dots, s_D^j)$ and $(d_1^j, d_2^j, \dots, d_D^j)$ is injected in the network using port I_u^{PE} **if and only if** $s_u^j \neq d_u^j$ and $\forall x \mid u < x \leq D, s_x^j = d_x^j$.

From Property 6, we get that all the packets of a given flow will be injected using the same input port. Note that, Property 6 is a generalization of Property 1.

The ports $O_1, O_2, \dots, O_D$ of a router are connected to the ports $I_1, I_2, \dots, I_D$ of its neighboring routers, but also serve as inputs to the programming elements. That is, the programming element connected to a router can read packets from all the output ports $O_1, O_2, \dots, O_D$. We show this property (i.e., that the programming element has read-access to all output ports of the router) using the notations $O_1^{PE}, O_2^{PE}, \dots, O_D^{PE}$ in Figure 7.2. We assume that a programming element can read packets from several different output ports simultaneously. This may be done by considering that each programming element has a FIFO queue connected to each port O_u^{PE} (with $1 \leq u \leq D$). We assume that those FIFO queues are large enough to prevent back pressure in the network. Although this design solution may lead to increased router programming logic complexity, it avoids the extra cost of implementing expensive exit multiplexers.

We consider that each programming element has also a FIFO queue connected to each port I_u^{PE} . These queues store flits that are pending to be injected in the network. Note that, the FIFO queues connected to each O_u^{PE} and I_u^{PE} port could be implemented in software or hardware. Their specific implementation is irrelevant to the matter discussed in this work.

We assume that no traffic injection regulator exists at the programming elements. Therefore, they can inject flits into the network as fast as possible. Nonetheless, we assume that each flow

f_j can have a maximum of one packet in the FIFO queue pending to be injected in the network at any moment in time. That is, only after a packet is injected, a new packet from the same flow f_j can be stored in the FIFO queue. The implicit assumption is that the minimum inter-arrival time T_j between the generation of every two packets of f_j is larger or equal to the worst-case packet injection time $wcit_j$ of that flow, i.e., $\forall f_j, T_j \geq wcit_j$. Note that, the restriction is only related to the content of the FIFO queues at the injection ports and does not limit the number of in-flight packets in the network. That is to say, several packets from the same flow f_j can be traveling around the network at the same time. Also, note that this assumption is less constraining than those made in many works on real-time NoCs that assume periods larger than the worst-case communication time (of which the injection time is just one component).

7.3.3 Routing Policy

Consider a flit that must travel from router (0;0;0) to router (2;0;0) in the example network of Figure 7.1a. It will reach its destination faster if it travels on the green link than if it hops through the red or blue links. Since the green, red, and blue links correspond to dimensions $\vec{D}_1, \vec{D}_2$, and $\vec{D}_3$, respectively, it is equivalent to say that it is faster for the flit to travel on a dimension of lower order. nDimNoC's routing policy simply builds upon that property. Additionally, it uses the idea of deflection routing [5] to avoid the cost of packet buffering. The approach is as follows.

Consider a flit of a flow f_j that has been injected at the origin router $(s_1^j, s_2^j, \dots, s_D^j)$ and with destination $(d_1^j, d_2^j, \dots, d_D^j)$. As mentioned in Section 7.3.2, the programming element injects that flit on port I_u^{PE} such that $s_u^j \neq d_u^j$ and $\forall x \mid u < x \leq D, s_x^j = d_x^j$.

If the flit was transmitted in isolation (i.e., without any interfering flow), it would travel along the dimension $\vec{D}_u$ of the network by entering in each router by input port I_u and requesting output port O_u . Then, when it reaches the first router R_k with the same coordinates $r_2^k, r_3^k, \dots, r_D^k$ as its destination (i.e., $r_b^k = d_b^j, \forall b \in [2, D]$), it would request the output port O_1^k and travel along dimension $\vec{D}_1$ until reaching its destination. It results that flits entering by input port I_u (such that $2 \leq u \leq D$) may only request the output port O_u or O_1 . Flits entering by the input port I_1 may only request the output port O_1 .

If there is interfering traffic, nDimNoC's routing policy allows flits to be deflected to make place for higher-priority traffic. Two such scenarios may happen:

1. If multiple flits entering by different input ports request the output port O_1 at the same time, nDimNoC always gives the highest priority to the flit that entered by the input port with the highest dimension (i.e., I_D wins over I_{D-1} , which wins over I_{D-2} , etc.). Consider two flits entering by ports I_u and I_v such that $u < v$ and that request output port O_1 . Then, the flit entering by I_v exists through O_1 , and the flit entering by I_u exists through O_{u+1} . We say that the flit that entered by I_u is deflected to dimension $\vec{D}_{u+1}$.
2. A flit entering by port I_u that was deflected to the output port O_{u+1} may now conflict for port O_{u+1} with a flit coming from I_{u+1} and that requests O_{u+1} at the same time. Under this

contention scenario, the flit coming from I_u and that was deflected towards O_{u+1} wins the right to use O_{u+1} and the flit coming from I_{u+1} is deflected towards the output port O_{u+2} .

Note that deflections redirect deflected flits on longer paths toward their destination. However, the topology presented in Section 7.3.1 ensures that it still progresses towards its destination router. Therefore, nDimNoC's routing policy is deadlock-free and livelock-free. Furthermore, after each deflection, a flit's priority to request output port O_1 in a future router increases (since flits traveling on higher dimensions have higher priorities). Therefore, its probability of being able to later travel on a shorter route increases too.

Finally, flits injected by the programming element (i.e., flits entering by any port I_u^{PE}), always have the lowest priority and must wait for the respective port O_u to be free. Table 7.1 summarizes the routing policy of a D-dimensional nDimNoC.

Table 7.1: Generic routing policy table of nDimNoC with D dimensions

Rule	Flows requests	Conflicting requests	Routing decisions	Explanation
1	$I_D \rightarrow O_D$	None	$I_D \rightarrow O_D$	No contention over O_D .
2	$I_D \rightarrow O_1$	Any	$I_D \rightarrow O_1$	$I_D \rightarrow O_1$ always wins.
3	$I_u \rightarrow O_u$	None	$I_u \rightarrow O_u$	No contention over O_u .
4		I_{u-1} deflected to O_u	$I_u \rightarrow O_{u+1}$	Flows coming from the I_{u-1} and I_u ports conflict over O_u . $I_{u-1} \rightarrow O_u$ always wins over $I_u \rightarrow O_u$. The flow coming from the I_u port is deflected to the O_{u+1} port.
5	$I_u \rightarrow O_1$	None or $I_{v < u} \rightarrow O_1$	$I_u \rightarrow O_1$	No flow entering by a port on a higher dimension than I_u requests O_1 . $I_u \rightarrow O_1$ wins.
6		$I_{v > u}$	$I_u \rightarrow O_{u+1}$	A flow entering by a port on a higher dimension than I_u wins O_1 . The flow coming from the I_u port is deflected to the O_{u+1} port.
7	$I_u^{PE} \rightarrow O_u$	None	$I_u^{PE} \rightarrow O_u$	There is no flow coming from another port that requests O_u . The flow on I_u^{PE} is injected in the network via O_u .
8		$I_v \rightarrow O_u$ and/or I_{u-1} deflected to O_u	None	The flow waiting on the I_u^{PE} port conflicts over the O_u port with flows coming from neighboring routers. Since flows from I_u^{PE} have the lowest priority, the flow waiting on the I_u^{PE} port is not injected in the network.

Example. Consider a 4x2x2 3-dimensional nDimNoC (i.e., $D = 3$) (see 7.3a- 7.3d). Each 3D-nDimNoC router has six input ports ($I_1, I_2, I_3, I_1^{PE}, I_2^{PE}$, and I_3^{PE}) and three output ports (O_1, O_2 , and O_3). Consider also a flit of a flow f_j (yellow flit in 7.3a) with origin and destination coordinates $(0;0;1)$ and $(3;1;0)$, respectively. Since $s_3^j \neq d_3^j$, the flit is injected via input port I_3^{PE} (see 7.3a). The flit then travels along the dimension $\vec{D}_3$ until it reaches router R_k with the same coordinates $r_2^k, r_3^k, \dots, r_D^k$ as its destination, i.e., $r_2^k = d_2^j = 1$ and $r_3^k = d_3^j = 0$ (see 7.3a). In R_k , the

flit enters by input port I_3 and requests output port O_1 to travel along the dimension $\vec{D}_1$ until its destination (see 7.3b). According to rule 2 of nDimNoC's routing policy (see Table 7.1), it has the highest priority to use O_1 and therefore enters the router $(1; 1; 0)$ (next router to R_k on dimension $\vec{D}_1$) by its port I_1 , and requests port O_1 (see 7.3b). If a flit enters by the input I_2 (blue flit in 7.3b) and/or I_3 port (pink flit in 7.3b) and request O_1 at the same time as the yellow flit, then the yellow flit is deflected to the output port O_2 (see 7.3c and rule 6 in Table 7.1). Thus, it must now travel along dimension $\vec{D}_2$ until it reaches the same router as it would have if it could have used the O_1 port instead. Note that the yellow flit may still suffer additional deflections to dimension $\vec{D}_3$ in any router it reaches while traveling along dimension $\vec{D}_2$ as it is the case on 7.3c where both a flit entering by the I_1 port (violet flit) and a flit entering by the I_3 port (orange flit) request the O_1 port. Then, the request $I_3 \rightarrow O_1$ wins over the other requests and the flits entering by the I_1 and I_2 ports are deflected to the O_2 and O_3 ports, respectively (see Figure 7.3d and rule 4 in Table 7.1).

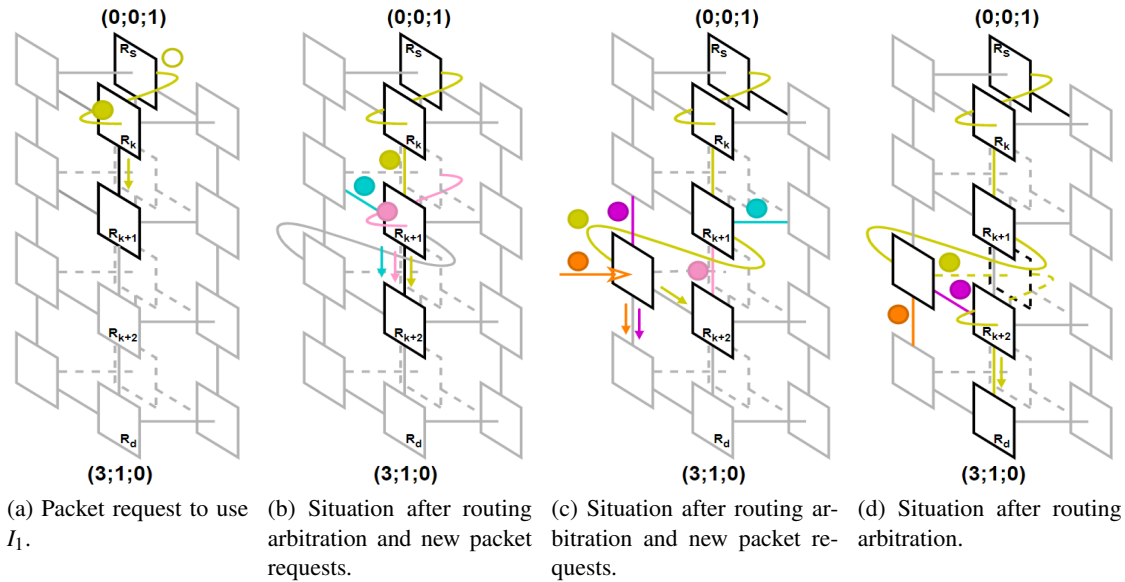


Figure 7.3: Example of nDimNoc's routing policy.

7.4 Worst-Case Communication Time Analysis

In section 7.3, we presented nDimNoC's design. In this section, we present an analysis of the worst-case communication time (WCCT) between two PEs connected with nDimNoC. We introduce a novel approach to compute bounds on traversal times by modeling the potential routes flits may take through all the routers using a directed acyclic graph.

7.4.1 Best- and Worst-Case Traversal Time Analysis

In this section, we compute bounds on the worst- and best-case traversal time of a flit p (abbreviated WCTT and BCTT, respectively). A bound on the WCIT is later derived in Section 7.4.2.

As discussed in the previous section, a flit p of flow f_j that travels through nDimNoC can be deflected in any router on its path to its destination, but there is only a limited set of routers in which it can *actively* request to change the dimension it travels along. Those routers are (i) the origin router of the flit with coordinates $(s_1^j, s_2^j, \dots, s_D^j)$, and (ii) every router R_k on the path of p such that its coordinates respect $r_b^k = d_b^j, \forall b \in [2, D]$. We formally denote this set of routers by $\mathcal{R}$ where

$$\mathcal{R} = \{R_k \mid \forall l \in [1, D], r_l^k = s_l^j \vee \forall b \in [2, D], r_b^k = d_b^j\}. \quad (7.4)$$

Note that the destination router of flow f_j is obviously in $\mathcal{R}$ since that router has the coordinates $(d_1^j, d_2^j, \dots, d_D^j)$.

As will be shown later in this section, the routing decisions in the routers in $\mathcal{R}$ are the only ones that must be analyzed to get a bound on the BCTT and WCTT of a flit p .

We use a directed acyclic graph (DAG) G modeling all the routes that a flit p may take to compute the WCTT and BCTT of a flit of a flow f_j that traverses a D -dimensional nDimNoC. A DAG $G = (V, E)$ is formed by a set of vertices V and a set of edges E . Each edge $e \in E$ connects two vertices u and v in E . We note $e = (u, v)$. Each edge is assigned a weight $w(u, v)$.

The DAG compactly represents all the routes that the flit p may potentially follow (from its origin to its destination) when it traverses nDimNoC. Each vertex v in the DAG G represents one input port of a router in $\mathcal{R}$. Let $R(u)$ and $I(u)$ be the router and the input port of the router represented by vertex u in the graph, then we note $u = (R(u), I(u))$. Each edge $e = (u, v) \in E$ connecting vertices u and v represents a possible path taken by the flit from input port $I(u)$ of router $R(u)$ to the input port $I(v)$ of another router $R(v)$ on its path. The weight of the edge $e = (u, v)$ is the maximum number of hops from $I(u)$ to $I(v)$ according to that path. Additionally, we label each edge $e = (u, v)$ with the specific output port taken by the flit in router $R(u)$.

Example. Figure 7.4 shows the DAG of the example of Section 7.3.3 (see Figure 7.3). It shows the potential paths that a flit of a flow f_j may follow from the origin router $(0; 0; 1)$ to the destination router $(3; 1; 0)$ when it traverses a $4 \times 2 \times 2$ 3-dimensional nDimNoC. The source vertex v_0 at level 0 of the graph represents the input port I_3^{PE} of the origin router R_s at which the flit is injected by the programming element. Since the flit p is injected by I_3^{PE} , it can only exist by output port O_3 of R_s . R_k is the first router p reaches after leaving R_s where p may request output port O_1 . Flit p may only enter R_k by the input port I_3 . Vertex v_1 on Level 1 represents input port I_3 of R_k . The weight w_1 is the number of hops the flit p does from the O_3 port of R_s to the I_3 port of R_k . In router R_k , the flit enters by the I_3 port and requests the O_1 port to travel along dimension $\vec{D}_1$. According to rule 2 of nDimNoC's routing policy (see Table 7.1), the routing decision for that request is always $I_3 \rightarrow O_1$ (because any flit entering by the I_3 port has the highest priority to use the O_1 port in a 3-dimensional nDimNoC). Therefore, p reaches router R_{k+1} in one hop, and certainly enters R_{k+1} by port I_1 . Since R_{k+1} is also in $\mathcal{R}$, it is represented by vertex v_2 . According to Table 7.1, two different routing decisions may be taken in R_{k+1} : (1) p is routed to the O_1 port if there is no conflict over O_1 (see rule 5 in Table 7.1); or (2) p is deflected to the O_2 port if there is one or more flows coming from other ports that request the O_1 port at the same time as p (see rule

6 in Table 7.1). If the situation (1) happens (i.e., the flit under analysis is routed to the O_1 port), it enters the router R_{k+2} using port I_1 . We represent the I_1 port of R_{k+2} as vertex v_3 in Level 3. If situation (2) occurs (i.e., the flit is deflected to the O_2 port), it may enter router R_{k+2} from: (1) the I_2 port if it suffer no further deflection to reaching R_{k+2} (see rule 3 in Table 7.1) or (2) the I_3 port if it suffers more deflections on its path to R_{k+2} (see rule 4 in Table 7.1). We represent the I_2 and I_3 ports of R_{k+2} as vertices v_4 and v_5 in the level 3 of the graph, respectively. Considering the potential routing decisions to which the flit p may be subjected after it enters R_{k+2} by the ports I_1 , I_2 or I_3 , p may reach its destination router R_d by input ports I_1 , I_2 or I_3 (vertices v_6 , v_7 , and v_8 on Level 4) in a maximum number of hops represented by the weights of the edges connecting the vertices of level 3 to those of level 4. Note that, the flit will always be received by the programming element regardless of the routing decision taken in the destination router.

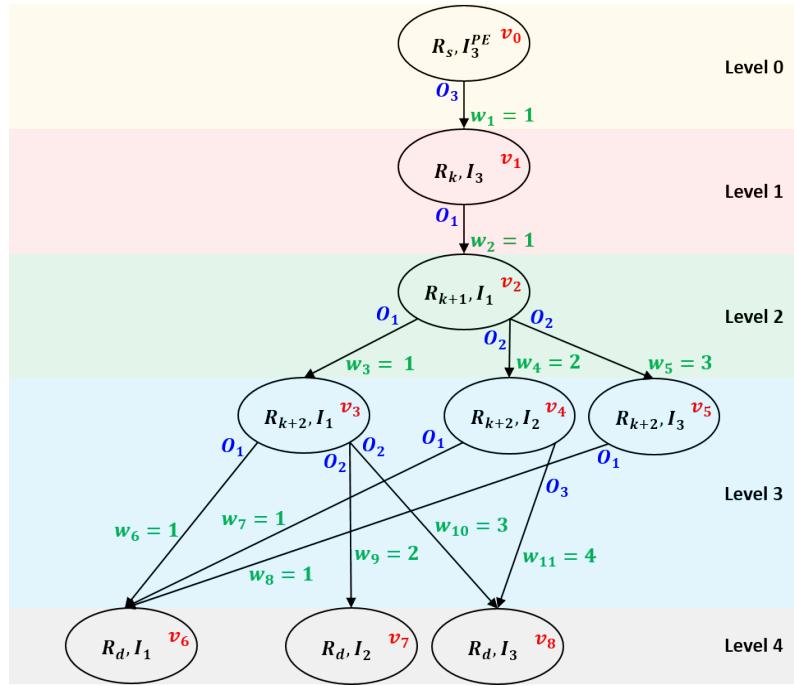


Figure 7.4: DAG of the potential trajectories that a flit may take from the origin $(0;0;1)$ to the destination $(3;1;0)$ when it traverses a $4 \times 2 \times 2$ 3-D nDimNoC.

After building the graph G as exemplified above, the WCTT of flit p is the longest weighted path in graph G , and its BCTT is the shortest weighted path in G . For the example of 7.4, the WCTT is thus equal to 8 and corresponds to the case where the flit p follows the path represented by vertices v_0 , v_1 , v_2 , v_4 and v_8 . Similarly, taking the shortest weighted path, we get that the BCTT of p is 4 in that example. Note that the WCTT may not always be obtained when the flit experiences its maximum number of deflections, hence the need for building the full graph G .

Following the reasoning above, the graph G can systematically be built using Algorithm 1. Algorithm 1 uses Lemmas 27 to 31 to compute the set of input and output ports to which the flit p

may be routed in each router in $\mathcal{R}$, and to compute the weight of each edge. We now present and prove those lemmas.

Algorithm 1: Building the DAG of the potential trajectories.

```

Input: flow  $f_j$ ;
Output:  $V, E$ ;
1  $V \leftarrow \emptyset; E \leftarrow \emptyset$ ;
2 Build set  $\mathcal{R}$  according to Equation 7.4;
3  $R_{cur} \leftarrow$  source router of  $f_j$ ;
4  $I_u \leftarrow$  input port by which  $f_j$  is injected in its source router according to 6;
5 Create vertex  $v_{cur} = (R_{cur}, I_u)$ ;
6  $V \leftarrow V \cup \{v_{cur}\}$ ;
7  $\Gamma^I \leftarrow \{I_u\}$ ;
8  $\Gamma_{new}^I \leftarrow \emptyset$ ;
9 while  $R_{cur}$  is not the destination router of  $f_j$  do
10    $R_{next} \leftarrow$  first router in  $\mathcal{R}$  reached by any flit of  $f_j$  after it leaves  $R_{cur}$ ;
11   foreach  $I_{cur} \in \Gamma^I$  do
12      $v_{cur} \leftarrow$  get vertex  $(R_{cur}, I_{cur})$  in  $V$ ;
13      $\Gamma^O \leftarrow$  Set of output ports to which the flit may be routed if it enters  $R_{cur}$  by the input port
        $I_{cur}$ ; // use Lemmas 27 and 28
14     foreach  $O_{cur} \in \Gamma^O$  do
15        $\Gamma_{next}^I \leftarrow$  Set of input ports by which the flit may enter  $R_{next}$  if it exits  $R_{cur}$  by the
         output port  $O_{cur}$ ; // use Lemmas 29 and 30
16       foreach  $I_{next} \in \Gamma_{next}^I$  do
17         if  $I_{next} \notin \Gamma_{new}^I$  then
18            $\Gamma_{new}^I \leftarrow \Gamma_{new}^I \cup \{I_{next}\}$ ;
19           Create vertex  $v_{next} = (R_{next}, I_{next})$ ;
20            $V \leftarrow V \cup \{v_{next}\}$ ;
21         else
22            $v_{next} \leftarrow$  get vertex  $(R_{next}, I_{next})$  in  $V$ ;
23         end
24         Create edge  $e = (v_{cur}, v_{next})$  with weight  $h_{O_{cur} \rightarrow I_{next}}^{R_{cur} \rightarrow R_{next}}$ ; // use Lemma 31
25          $E \leftarrow E \cup \{e\}$ ;
26       end
27     end
28   end
29    $\Gamma^I \leftarrow \Gamma_{new}^I$ ;
30    $\Gamma_{new}^I \leftarrow \emptyset$ ;
31    $R_{cur} \leftarrow R_{next}$ ;
32 end

```

In the following, we denote by R_{cur} and R_{next} any two routers in $\mathcal{R}$ such that R_{next} is the first router in $\mathcal{R}$ reached by p after leaving R_{cur} .

Lemma 27. A flit p of a flow f_j that enters router R_{cur} by port I_u^{PE} will be routed to the output port O_u .

Proof. By rule 7 of nDimNoC's routing policy (see Table 7.1). □

Lemma 28. A flit p that enters router R_{cur} by port I_u may be routed to any of the output ports belonging to the set Γ_u^O , such that

$$\Gamma_u^O = \begin{cases} \{O_1\} & \text{when } u = D \\ \{O_1\} \cup \{O_{u+1}\} & \text{when } u \neq D \end{cases} \quad (7.5)$$

Proof. According to rule 2 in Table 7.1, a flit entering by the I_D port has the highest priority to use the O_1 port and will never be deflected to any other output port. This proves the first case of Equation 7.5. If the flit enters the router by an input port I_u such that $u < D$ and requests output port O_1 , two scenarios may happen according to Table 7.1: (1) it wins port O_1 (see rule 5 in Table 7.1), or (2) it is deflected to port O_{u+1} (see rule 6 in Table 7.1). This proves the second case of Equation 7.5. $\square$

Lemma 29. Let O_u be the output port by which flit p exits the router R_{cur} . If R_{next} is only one hop further on dimension $\overrightarrow{D_u}$, then flit p enters R_{next} by its port I_u .

Proof. Since, according to the network topology defined in 7.3.1, the output port O_u of R_{cur} is connected to the input port I_u of R_{next} , and because flit p exits R_{cur} by port O_u , the lemma follows. $\square$

Lemma 30. Let O_u be the output port by which flit p exits the router R_{cur} . If R_{next} is more than one hop away from R_{cur} on dimension $\overrightarrow{D_u}$, then the flit p will enter R_{next} by one of the input ports belonging to the set Γ_u^I , such that,

$$\Gamma_u^I = \{I_v \mid u \leq v \leq D\} \quad (7.6)$$

Proof. If R_{next} is more than one hop away from R_{cur} on dimension $\overrightarrow{D_u}$, flit p must hop through at least one other router between R_{cur} and R_{next} . First, we note that by definition, R_{next} is the first router after R_{cur} on flit p 's path where p may request output port O_1 . Thus, according to nDimNoC's routing policy (7.3.3), p may only continue to travel along the same dimension (see rule 3 in Table 7.1) or be deflected to a higher order dimension while traveling between R_{cur} and R_{next} (see rule 4 in Table 7.1).

If no deflection happens in the routers located between R_{cur} and R_{next} , flit p will enter R_{next} by input port I_u . However, according to rule 4 of nDimNoC's routing policy (see Table 7.1), if $u < D$, the flit p may also be deflected to dimension $\overrightarrow{D_{u+1}}$ in one of those intermediate routers. If no further deflection happens until reaching R_{next} , p will then enter R_{next} by the input port I_{u+1} . Yet, if $u+1 < D$, Table 7.1 states that the flit p may still be deflected to dimension $\overrightarrow{D_{u+2}}$ while traveling along dimension $\overrightarrow{D_{u+1}}$. Repeating this reasoning, we get that flit p may enter R_{next} by any input port I_v such that $u \leq v \leq D$. $\square$

Lemma 31. The maximum number of hops from the output port O_u of R_{cur} to the input port I_v of R_{next} (with $u \leq v$) is upper bounded by

$$h_{O_u \rightarrow I_v}^{R_{cur} \rightarrow R_{next}} = (v - u) + \frac{(pos^{next} - pos^{cur'} + N) \bmod N}{g_{D-v+1}} \quad (7.7)$$

where

$$pos^{cur'} = (pos^{cur} + \sum_{k=u}^{v-1} g_{D-k+1}) \quad (7.8)$$

and pos^{cur} and pos^{next} are computed with Equation 7.1.

Proof. According to nDimNoC's routing policy and following the same explanation as in the proof of Lemma 30, a flit that exits R_{cur} by port O_u and enters R_{next} by port I_v must have been deflected exactly $(v - u)$ times.

According to 5, flit p bypasses g_{D-k+1} routers on the main ring of the network with every hop it does on dimension $\vec{D}_k$. Because, by definition of our circulant topology, we have $g_{D-k+1} > g_{D-k}$ for all k , the flit p will make its maximum number of hops when it suffers its $(v - u)$ deflections as early as possible and thus travels as long as possible along the highest order dimension, i.e., along $\vec{D}_v$.

In such scenario, the flit does exactly one hop on each dimension $\vec{D}_u, \vec{D}_{u+1}, \vec{D}_{u+2}, \dots, \vec{D}_{v-1}$ and bypasses $\sum_{k=u}^{v-1} g_{D-k+1}$ routers on the network's main ring. Thus, after the $(v - u)$ initial hops, the flit reaches router $R_{cur'}$ situated $\sum_{k=u}^{v-1} g_{D-k+1}$ steps further than R_{cur} on the main ring. That is, the position of $R_{cur'}$ on the main ring is given by Equation 7.8.

Since the network contains N routers on its main ring, the router $R_{cur'}$ and R_{next} are still $(pos^{next} - pos^{cur'} + N) \bmod N$ steps away from each other on that ring. However, since the flit p only travels along dimension $\vec{D}_v$ after it reaches $R_{cur'}$, it bypasses g_{D-v+1} routers of the main ring at each hop. Thus, it needs $\frac{(pos^{next} - pos^{cur'} + N) \bmod N}{g_{D-v+1}}$ hops from port O_v of $R_{cur'}$ to port I_v of R_{next} . Therefore, in total, flit p does $(v - u)$ hops to reach $R_{cur'}$ and $\frac{(pos^{next} - pos^{cur'} + N) \bmod N}{g_{D-v+1}}$ additional hops to reach R_{next} , hence proving Equation 7.7. $\square$

Corollary 2. If $u = v$, then $h_{O_u \rightarrow I_v}^{R_{cur} \rightarrow R_{next}}$ is an exact bound on the number of hops between the output port O_u of R_{cur} and the input port I_v of R_{next} .

Proof. According to nDimNoC's routing policy, any deflection of a flit p between R_{cur} and R_{next} would result in p entering R_{next} by an input port I_v such that $v > u$. Therefore, if $u = v$, flit p must not have suffered any deflection and must have traveled along dimension $\vec{D}_u$ only. Because $(pos^{next} - pos^{cur} + N) \bmod N$ is the distance between R_{cur} and R_{next} on the main ring of the network, and because for every hop on dimension $\vec{D}_u$, packet p bypasses g_{D-u+1} routers on the main ring (by 5), we have that p reaches R_{next} in exactly $\frac{(pos^{next} - pos^{cur} + N) \bmod N}{g_{D-u+1}}$ hops. Note that this last equation is equal to $h_{O_u \rightarrow I_v}^{R_{cur} \rightarrow R_{next}}$ when $v = u$, which proves the claim. $\square$

We now prove that the WCTT and BCTT of a flit of flow f_j are bounded by the longest and the shortest weighted path of the graph G returned by Algorithm 1, respectively. To do so, we first

prove that the graph G built using Algorithm 1 contains all routes that may be taken by packets of flow f_j between its origin and destination.

Lemma 32. The DAG built using Algorithm 1 contains one edge for each possible path between any two routers in $\mathcal{R}$ that may be successively traversed by any flit of flow f_j .

Proof. Algorithm 1 iterates over all routers in $\mathcal{R}$ that are on the path of f_j from its origin to its destination router (Lines 3, 9, 10, and 31). For each pair of routers R_{cur} , R_{next} , the algorithm computes the set Γ^I of all input ports by which f_j may enter R_{cur} . For each such input, it uses Lemmas 27 and 28 to compute the set Γ^O of all output ports by which f_j may exit R_{cur} (Line 13). For each output port $O_{cur} \in \Gamma^O$, it then uses Lemmas 29 and 30 to compute the set Γ_{next}^I of all input ports by which f_j may enter R_{next} (Line 15). It finally creates an edge for every path between O_{cur} and the input ports in Γ_{next}^I (Line 24). Since Lemmas 27 to 30 were all proven correct, we have that Algorithm 1 creates an edge for every possible path between any two routers R_{cur} and R_{next} in $\mathcal{R}$, i.e., one edge for any combination of output and input port of R_{cur} and R_{next} that may be successively traversed by a packet of f_j . $\square$

Lemma 32 has the following corollary as a direct consequence.

Corollary 3. The DAG built using Algorithm 1 contains all possible paths taken by flow f_j from its origin to its destination router.

Theorem 3. The longest weighted path of the DAG built with Algorithm 1 is an upper bound on the WCTT of any flit of flow f_j .

Proof. By Lemma 32 and Corollary 3, the DAG built with Algorithm 1 contains all possible routes from the origin to the destination of f_j encoded as a different path in the graph. Furthermore, by Lemma 31 and Line 24 of Algorithm 1, the weight of every edge in the graph is an upper bound on the number of hops on the longest path between the output and input ports of the two routers represented by the vertices connected by that edge. Thus, the longest weighted path in the graph is an upper bound on the number of hops between all routers on the path of f_j from its origin to its destination. This proves the Theorem. $\square$

Theorem 4. The shortest weighted path of the DAG built with Algorithm 1 is the BCTT of any flit of flow f_j .

Proof. According to nDimNoC's routing policy and its discussion in Section 7.3.3, a flit p of flow f_j performs its minimum number of hops between its origin and destination router when it does not suffer any deflection.

Since the DAG built with Algorithm 1 contains all possible routes from the origin to the destination of f_j encoded as a different path in the graph (by Lemma 32 and Corollary 3), it also contains the path where the flit of f_j does not suffer any deflection. Furthermore, by Corollary 2 and Line 24 of Algorithm 1, the weight of every edge corresponding to a path where p does not suffer deflection is equal to the exact number of hops performed by p on that path. Therefore,

the shortest weighted path in the graph is an exact bound on the BCTT of f_j from its origin to its destination. This proves the Theorem. $\square$

7.4.2 Bound on the Worst-Case Injection Time

In the previous section, we explained how to compute bounds on the BCTT and WCTT of any flit of a packet injected by a flow f_j . In this section, we derive a bound on the worst-case injection time WCIT of any packet of f_j (see Theorem 5).

First, we recall a bound on the maximum number of flits that may be injected in the network by any flow f_j . In accordance with Lemma 14 of Section 3.4.3, in any time interval of length Δt , the flow f_j can inject in the network at most $\lambda_j(\Delta t) = \min \left\{ \Delta t, \left\lceil \frac{\Delta t + wcit_j}{T_j} \right\rceil C_j \right\}$ flits.

Then, we prove an upper bound on the WCIT of any packet of f_j using Lemma 25. To prove that theorem, we use the following notation: flow f_j is injected in router R_{inj} via input port I_{inj}^{PE} (i.e., R_{inj} is the origin router of f_j). We define $\mathcal{F}_{inj}$ as the set of all flows (including f_j) injected in the same input port I_{inj}^{PE} of the same router R_{inj} as f_j . Note that this set of flows is a property of the system and thus we assume that it is given as an input to the analysis. We also define Γ_{inj}^{conf} as the set of all flows originating from other routers than R_{inj} and that may conflict with the injection of flow f_j in router R_{inj} . The content of Γ_{inj}^{conf} is computed using Lemmas 33 and 37 proven later in this section.

Theorem 5. The WCIT $wcit_j$ of any packet of flow f_j is given by the smallest positive solution to the recursive equation

$$wcit_j \geq \left(\sum_{\forall f_i \in \mathcal{F}_{inj}} C_i \right) - 2 + \sum_{\forall f_l \in \Gamma_{inj}^{conf}} \lambda_l(wcit_j + 1 + J_l) \quad (7.9)$$

where $J_l = wctt'_l - bctt'_l$ is the difference between the worst-case and the best-case traversal time of flow f_l until router R_{inj} (computed with Theorems 3 and 4).

Proof. Let p be the last flit of any packet of flow f_j . According to nDimNoC's routing policy, the flit p will be injected in the network as soon as: 1) all flits ahead of p in the FIFO queue of the input port by which it is injected in the network have been injected, and 2) there is one clock cycle during which no packet entering R_{inj} from other input ports conflicts for the same output port as p .

Let n_{flits} be the maximum number of flits ahead of p in the FIFO queue, and let $W_u(\Delta t)$ be the maximum number of flits entering into R_{inj} by another input port than p and requesting the same output port as p in a time interval of length Δt . Then, conditions 1) and 2) are met as soon as

$$\Delta t + 1 \geq n_{flits} + W_u(\Delta t + 1) \quad (7.10)$$

for $\Delta t \geq 0$. The solution to Equation 7.10 is thus equal to the WCIT $wcit_j$ of flow f_j .

To solve the above equation, we first derive a bound on n_{flits} , and then derive a bound on $W_u(\Delta t + 1)$.

Bound on n_{flits} : Section 7.3.2 explains that each flow in $\mathcal{F}_{inj}$ may have at most one packet in the FIFO queue of the input port by which they are injected in the network. Therefore, in the worst-case scenario, there is one packet of each other flow injected via the same port I_{inj}^{PE} as f_j ahead of p in the FIFO queue. Since p is the last flit of f_j 's packet, there must also be $(C_j - 1)$ other flits of f_j ahead of p in the FIFO queue. That is,

$$n_{flits} \leq \left(\sum_{\forall f_i \in \mathcal{F}_{inj}} C_i \right) - 1 \quad (7.11)$$

Bound on $W_u(\Delta t + 1)$: Let f_l be a flow entering the router R_{inj} by another input port than p but requesting the same output port as p (i.e., $f_l \in \Gamma_{inj}^C$). In the best and worst case scenarios, a flit from f_l takes $bctt'_l$ and $wctt'_l$ clock cycles to reach R_{inj} , respectively. Therefore, the first flit of f_l that may conflict with the injection of p must have been injected no earlier than $wctt'_l$ clock cycles before the beginning of the period during which p is interfered with. Conversely, the last flit of f_l that may conflict with the injection of p must have been injected no later than $bctt'_l$ clock cycles before the end of the interference with p . Therefore, the length of the interval during which f_l may inject flits that conflict with the injection of p is

$$\Delta t + 1 + wctt'_l - bctt'_l = \Delta t + 1 + J_l. \quad (7.12)$$

Lemma 14 states that a flow f_l may inject at most $\lambda_l(\Delta t + 1 + J_l)$ flits in that time interval. Therefore, the total number of flits from all conflicting flows $f_l \in \Gamma_{inj}^C$ is upper bounded as

$$W_u(\Delta t + 1) \leq \sum_{\forall f_l \in \Gamma_{inj}^C} \lambda_l(\Delta t + 1 + J_l) \quad (7.13)$$

Injecting Equations 7.11, 7.13 in Equation 7.10, we prove the lemma. $\square$

Theorem 5 requires to know the set of all flows Γ_{inj}^C that may conflict with the injection of f_j in router R_{inj} . The content of that set depends on the specific output port requested by the packets of f_j . Lemmas 33 and 37 below provide a means to compute the content of Γ_{inj}^C when f_j request output port O_1 or O_u (with $u \neq 1$), respectively.

Lemma 33. The set of flows coming from other routers than R_{inj} and that may be routed to output port O_1 of R_{inj} is given by

$$\Gamma_{inj}^1 = \{f_l \mid \forall b \in [2, D], d_b^l = r_b^{inj} \wedge \text{dist}(R_{orig}(f_l), R_{inj}) \leq \text{dist}(R_{orig}(f_l), R_{dest}(f_l))\} \quad (7.14)$$

Proof. According to nDimNoC's routing policy, a flow f_l may request the output port O_1 of R_{inj} only if (i) f_l may hop through router R_{inj} , and (ii) $\forall b > 1, d_b^l = r_b^{inj}$. Condition (i) requires that R_{inj} is located between the origin and destination router of f_l , i.e., $\text{dist}(R_{orig}(f_l), R_{inj}) \leq \text{dist}(R_{orig}(f_l), R_{dest}(f_l))$. Combining both (i) and (ii) proves the lemma. $\square$

To compute the set Γ_{inj}^C for the case where f_j requests output port O_u (with $u \neq 1$), we must first prove some intermediate results using Lemmas 34 to 36.

To prove those lemmas, we define $\mathcal{F}_{inj}^u$ as the set of all flows that may enter router R_{inj} by input port I_u . That set can easily be built by checking all paths that may be taken by each flow in the network according to Table 7.1. All those that have at least one path in which they enter R_{inj} by input port I_u are then added to the set $\mathcal{F}_{inj}^u$.

Lemma 34. The set of flows that may enter R_{inj} by input port I_u and request output port O_u is given by $\Gamma_{inj}^{u \rightarrow u} = \mathcal{F}_{inj}^u \setminus \{f_l \mid \forall b \in [2, D], r_b^{inj} = d_b^l\}$

Proof. According to Table 7.1, a flow entering by input port I_u and requesting output port O_1 cannot be routed to output port O_u (only to O_1 or O_{u+1}) (see rules 2, 5, and 6 in Table 7.1). Therefore, the set of flows entering by I_u that may be routed to O_u is the set of flows that enters R_{inj} by I_u (i.e., $\mathcal{F}_{inj}^u$) minus those that request O_1 , i.e., all the flows f_l that have a destination such that $\forall b \in [2, D], r_b^{inj} = d_b^l$ (see routing policy explained in Section 7.3.3). This proves the lemma. $\square$

Lemma 35. Let def_q be a boolean equal to true if a deflection may happen in router R_q , and equal to false otherwise. Then, we have

$$\text{def}_q = \begin{cases} \text{true} & \text{if } \exists u, v \text{ with } u \neq v \mid \exists f_l \in \mathcal{F}_q^u, \exists f_m \in \mathcal{F}_q^v \text{ s.t. } l \neq m \wedge \forall b \in [2, D], d_b^l = d_b^m = r_b^q \\ \text{false} & \text{otherwise.} \end{cases} \quad (7.15)$$

Proof. According to Table 7.1, a deflection may happen in router R_q only if at least two different flows compete to access the output port O_1 . For that situation to happen, there must exist at least two different flows f_l and f_m entering by two different input ports (i.e., $\exists u, v$ with $u \neq v \mid \exists f_l \in \mathcal{F}_q^u, \exists f_m \in \mathcal{F}_q^v$ s.t. $l \neq m$) that both request output port O_1 . According to the routing policy explained in 7.3.3, this happens only if $\forall b \in [2, D], d_b^l = d_b^m = r_b^q$. This proves the lemma. $\square$

Lemma 36. The set of flows that may enter R_{inj} by input port I_{u-1} and be routed to output port O_u is given by

$$\Gamma_{inj}^{u-1 \rightarrow u} = \begin{cases} \mathcal{F}_{inj}^{u-1} & \text{if } \text{def}_{inj} = \text{true} \\ \emptyset & \text{if } \text{def}_{inj} = \text{false} \end{cases} \quad (7.16)$$

Proof. According to Table 7.1, all flows that enter router R_{inj} by input port I_{u-1} (i.e., those in $\mathcal{F}_{inj}^{u-1}$) may be deflected to output port O_u (see rules 4 and 6 in Table 7.1). Thus, the set of flows entering by I_{u-1} and routed to O_u is given by all flows in $\mathcal{F}_{inj}^{u-1}$ if a deflection may happen in R_{inj} , i.e., if $\text{def}_{inj} = \text{true}$. If no deflection may happen in R_{inj} (i.e., $\text{def}_{inj} = \text{false}$), then Table 7.1 states that none of the flows entering by I_{u-1} may be routed to O_u . This proves both cases of Equation 7.16. $\square$

Lemma 37. The set of flows coming from other routers than R_{inj} and that may be routed to output port O_u of R_{inj} (with $u \neq 1$) is given by

$$\Gamma_{inj}^u = \Gamma_{inj}^{u \rightarrow u} \cup \Gamma_{inj}^{u-1 \rightarrow u} \quad (7.17)$$

Proof. According to Table 7.1, only flows that enter a router by its input ports I_u or I_{u-1} can be routed to output port O_u . Since, according to Lemmas 34,36, $\Gamma_{inj}^{u \rightarrow u}$ and $\Gamma_{inj}^{u-1 \rightarrow u}$ contain all the flows entering R_{inj} by input ports I_u and I_{u-1} , respectively, that may be routed to output port O_u , their union contains all flows that may come from other routers than R_{inj} and may be routed to output port O_u of R_{inj} . $\square$

The content of Γ_{inj}^C is thus equal to Γ_{inj}^1 if f_j requests output port O_1 (Lemma 33), and to Γ_{inj}^u if it requests any other output port (Lemma 37). Using Γ_{inj}^C in Lemma 25 we can now bound the WCIT of f_j .

7.5 Experimental Results

7.5.1 Implementation

We implemented a 3D-nDimNoC with the hardware description language Verilog. We synthesized a single router of 3D-nDimNoC for flits of 64 bits. The target platform was a Xilinx Kirtex-7 FPGA. It required 290 LUTs and 202 Flip-Flops (FFs) in total.

We compared the hardware cost of a 3D-nDimNoC with HopliteRT*, and with ProNoC, IDAMC, and CONNECT VCs-based NoCs. Table 7.2 shows the synthesis results. A 4x4x4 3D-nDimNoC requires 18,560 LUTs, i.e., between 3.9% to 28% of the Xilinx Kirtex-7 resources. That is, it is three times more expensive than HopliteRT* (which requires 5632 LUTs) but approximately 5-times cheaper than ProNoC, IDAMC, and CONNECT NoCs in terms of LUTs utilization. We thus conclude that 3D-nDimNoC is a suitable solution for FPGA platforms, as it leaves resources available for computation logic.

Table 7.2: Resources utilization of different NoCs in Kirtex-7 FPGAs.

NoC	LUTs	% Resource utilization of the platform
8x8 ProNoC	100000	20%-150%
8x8 IDAMC	83000	18%-127%
8x8 CONNECT	96000	20%-147%
8x8 HopliteRT*	5632	1.1%-8.5%
4x4x4 3D-nDimNoC	18560	3.9%-28%

Finally, we connected the nDimNoC router to a Microblaze soft-core and synthesized a 4x2x2 3D-network for a Virtex-7 485T using Xilinx Vivado. We computed the maximum operating frequency of the network with Xilinx Vivado. We obtained ≈ 210 MHz for a 4x2x2 3D-nDimNoC against ≈ 275 MHz for an equivalent 4x4 HopliteRT* NoC. This degradation in terms of maximum operating frequency may be explained by the fact that (1) a nDimNoC router requires more complex logic to route packets from its input to its output ports, and (2) the additional dimensions increase the number of wires between routers, which increases the complexity of the placement and routing during the logic synthesis.

7.5.2 NoC Latency Bounds

In this section, we provide experimental results by computing the WCTT, WCIT, and WCCT of sets of communication flows that traverse NoCs of different dimensionalities.

As a starting point, we generated sets of communication flows for a 16x16 2D-NoC according to a random traffic pattern. The origin and destination coordinates of each flow were randomly generated using a uniform probability distribution. The number of flits of packets released by a communication flow was randomly chosen between 1 and 5, and their inter-arrival times were generated as in [82]. Then, we made a one-to-one mapping of the routers in the 16x16 2D-NoC to the routers of a 4x8x8 3D-nDimNoC, a 4x4x4x4 4D-nDimNoC, a 2x2x4x4x4 5D-nDimNoC, and a 2x2x2x2x4x4 6D-nDimNoC. The origin and destination of each flow were accordingly updated for each network topology.

In Figures 7.5a and 7.5b, we show the maximum and average packets WCTT for an increasing number of flows in NoCs of different dimensionalities. The results were computed by using the analysis of HopliteRT and HopliteRT* (assuming a 16x16 2D-NoC), and the analysis presented in Section 7.4.1 for the 2D, 3D, 4D, 5D and 6D-nDimNoC topology. To establish a fair comparison, we assume one priority level (i.e., all flows were assigned the highest priority) for the analysis proposed for HopliteRT* in Section 3.4.1. Each point in the plot is the result of 100 repetitions (100 different random flow sets). We varied the number of generated flows from 10 to 300 by steps of 10.

In Figure 7.5a, we observe that the maximum WCTT is slightly worse with nDimNoC as compared to HopliteRT*. Yet, Figure 7.5b shows that the average traversal time improves with nDimNoC as the dimensionality of the network increases. This can easily be explained by the fact that new routes, possibly shorter and faster, are made available between pairs of routers when a new dimension is added to the network. Moreover, the number of interfering flows, and therefore, the number of deflections that flows may suffer on each link decreases since the number of routers on each dimension decreases. Note that, the average packets WCTT is reduced by $\approx 40\%$ and $\approx 60\%$ with a 5D-nDimNoC and a 6D-nDimNoC, respectively, against HopliteRT*. We also show that the maximum and average worst-case traversal times are noticeably reduced with nDimNoC as compared to HopliteRT.

In Figure 7.5c and 7.5d, we show the maximum and average WCIT of flows using the analysis of HopliteRT* and nDimNoC. We also computed the maximum and average WCIT by using the analysis of HopliteRT, but we do not show them on the graphs as they are extremely pessimistic and would render the plots unreadable by cluttering all other lines together. As shown, the packets see their WCIT drastically reduced in nDimNoC in comparison to HopliteRT*. This is expected since nDimNoC allows the PE connected to a router to inject packets simultaneously via as many input ports as there are dimensions in the network. A router of HopliteRT*, on the other hand, can inject at most one flit per cycle in the network (on either of the router output ports). Furthermore, the number of flows that may interfere with the injection of a packet at a router decreases since more routes are available in the network, and thus less traffic uses each individual route.

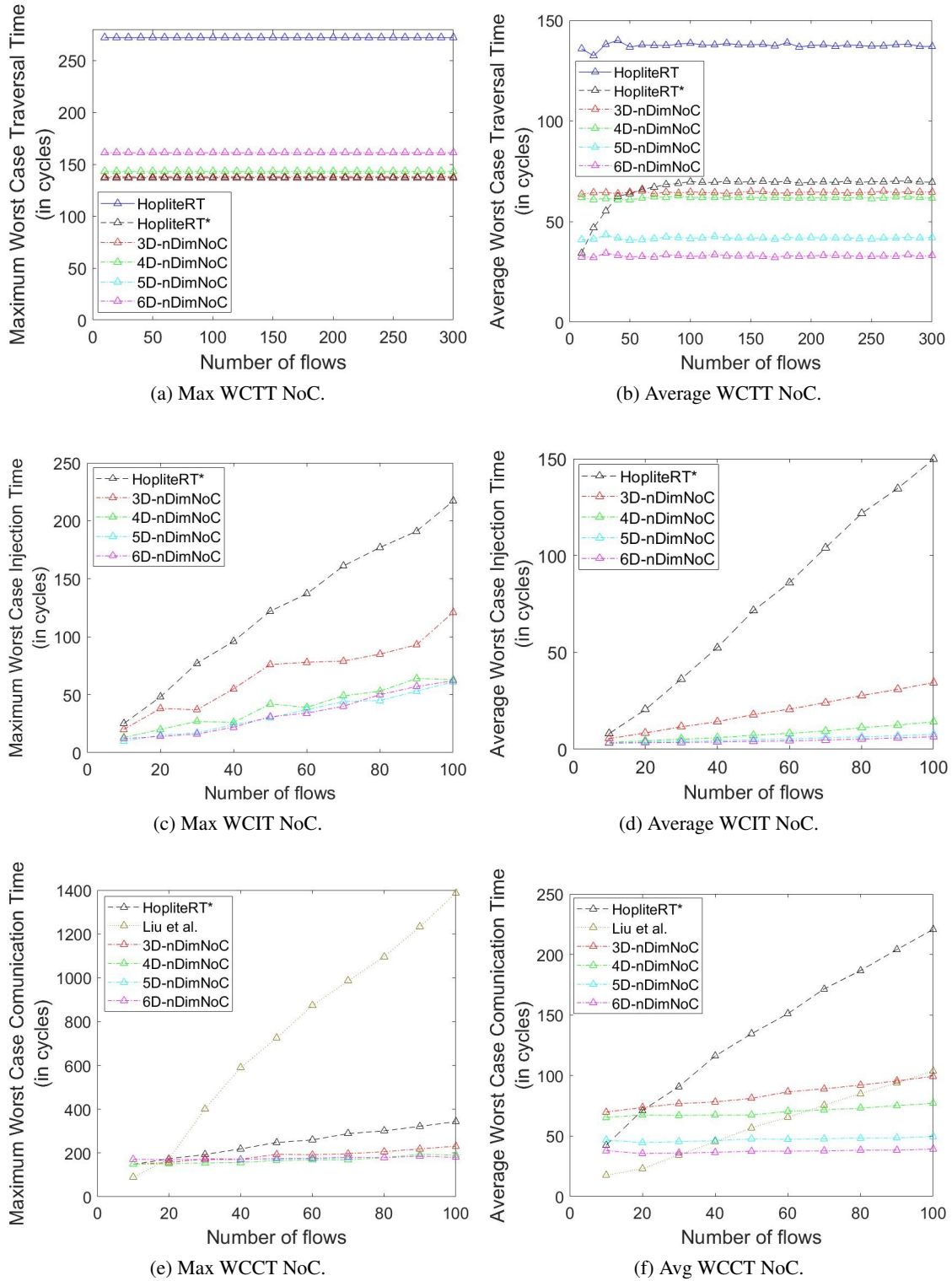


Figure 7.5: Experimental results for a random traffic pattern.

In Figure 7.5e and 7.5f, we show the maximum and average packets WCCT (which we recall to be equal to the sum of the WCTT and WCIT of those packets). We varied the number of

generated flows from 10 to 100 by steps of 10. The results were obtained by using the analysis of nDimNoC, and the analyses proposed in Section 3.4.1 and [53] for HopliteRT* and a VC-based real-time NoC, respectively. The analysis presented in [53] by Liu et al. is an improved analysis of that proposed in [82, 81] by Shi and Burns. To establish a fair comparison, we assume one VC (i.e., one priority level) for the analysis presented in [53]. As shown in 7.5e, for almost all configurations, the WCCT returned by the analysis of nDimNoC outperforms that returned by the analysis proposed in Section 3.4.1 and [53]. The average WCCT is only better with the analysis by Liu et al. when the network is completely underloaded and very few flows are traversing the network (i.e., less than 30 flows). Note also that, [53] considers that each flow may only have one packet of each flow traveling across the network at the same time, whilst nDimNoC supports the transmission of several packets from the same communication flow simultaneously.

The average WCCT with nDimNoC improves when the network's dimensionality increases and is barely impacted by the number of flows. Therefore, we conclude that increasing the dimensionality of nDimNoC has a positive impact from an average performance perspective for a limited impact on the worst-case performance of the flows.

7.6 Chapter Summary

In this chapter, we presented nDimNoC, an innovative and flexible real-time higher-dimensional NoC that leverages the advantages of circulant topologies to provide real-time guarantees, reduce average network communication latency, and offer greater flexibility compared to traditional 2D-NoCs. We employed DAG to compute bounds on the best- and worst-case traversal times of packets traversing NoCs with multiple dimensions. Additionally, we derived an upper-bound on the packets' WCIT, allowing for a comprehensive understanding of the timing behavior of nDimNoC NoC. Moreover, we provided a complete implementation of 3D-nDimNoC in HDL Verilog, enabling evaluation on hardware platforms. The experimental results demonstrated significant improvements in terms of network communication latency when compared to existing 2D solutions, confirming the efficacy of nDimNoC as a promising NoC solution for real-time applications.

Chapter 8

Thesis Summary, Conclusion and Future Work

In this chapter, we provide a summary of the contributions and outcomes achieved in this thesis, as well as identify potential avenues for future research. Specifically, the chapter is structured as follows. Section 8.1 offers an overview of our contributions and significant findings. Following that, in Section 8.2, we assess the validity of the thesis statement. Finally, in Section 8.3, we delve into prospective extensions and areas for future research, building upon the research presented in this thesis.

8.1 Summary of Contributions

The work carried out in this dissertation is divided into two parts: (1) designing and implementing NoC solutions and (2) deriving timing analyses for each of the proposed NoC architectures.

In Chapter 3, we introduced HopliteRT*, a novel buffer-less deflection-based real-time NoC featuring a priority-based routing policy. Leveraging a circulant topology, HopliteRT* optimizes the number of deflections suffered by flits, slightly increasing hardware resource usage compared to its predecessor, HopliteRT. Additionally, an issue in the timing analysis of HopliteRT was identified, and a formal derivation of a WCCT analysis for packets with different QoS requirements transmitted over HopliteRT* NoC was presented. Specifically for the WCCT, we developed two analyses. First, we proposed a timing analysis that considers only the analyzed packet's information, independent of other communication flows in the network. This analysis is suitable for dynamic systems where communication flows change over time, yet it can be pessimistic in certain cases. Subsequently, we derived a tighter and more accurate WCTT bound, particularly useful when additional system information is available.

In HopliteRT*, flits may not arrive at their destinations in the same order as they were injected at their sources due to the non-deterministic number of deflections that flits may experience while traveling through the network. In Chapter 4, we expanded upon the solution introduced in Chapter 3 by presenting IPDeN, a real-time deflection-based NoC that supports the in-order flit

delivery property. While avoiding the need for expensive buffers, our design centers on a single small buffer integrated into each router to prevent flits from overtaking others belonging to the same communication flow. The size of buffers in IPDeN is solely dependent on the network topology. This ensures that the implementation cost (in terms of hardware resource utilization) remains independent of the specific applications running on the network. Furthermore, we derived a WCTT analysis for packets transmitted over IPDeN, which is predictable and solely depends on the properties of the communication flow under analysis.

In IPDeN, during contention scenarios, flits are always forced to deflect toward alternative output ports while traveling through the network. Furthermore, to ensure in-order flit delivery at the destination, no deflected flits are frequently buffered within the routers. As a result, there is a potential increase in the average traversal time of flits, since each deflection and buffering comes with an associated cost. In Chapter 5, we introduced IPDeN 2.0, an enhanced version of IPDeN. This solution harmonizes the advantages of buffers with the concept of deflection to mitigate the necessity for continuous deflection of flits during contention scenarios. Specifically, it proposes a routing policy based on selective deflection and buffering of flits, thereby enhancing their traversal time while maintaining a bounded WCTT. The approach maintains the in-order flit delivery property introduced in IPDeN. We also proved that the upper-bounds on the WCTT of flits in IPDeN 2.0 remain unchanged from those in IPDeN.

In the context of real-time applications, the integration of a traffic injection regulation protocol into NoCs becomes crucial for the effective distribution of network bandwidth. This, in turn, enables the assurance of varying QoS levels for different communication flows. In Chapter 6, we introduced D-TSR protocol, which employs dynamic time slots requested by PEs that face challenges injecting their flits into the network due to congestion. This innovative strategy effectively manages traffic injection in NoCs based on network congestion. The protocol addresses in part the issue of certain PEs being blocked indefinitely due to unfair bandwidth utilization by other PEs. By strategically delaying certain flit injections into the network, we attain a balanced network bandwidth allocation across PEs. Additionally, we derived a novel upper-bound on the WCIT of flits when traffic injection is regulated by D-TSR protocol.

In Chapter 7, we introduced a novel buffer-less router architecture facilitating the synthesis of multi-dimensional NoCs. This design is accompanied by a deflection-based routing policy that relies on circulant topologies to provide real-time guarantees. A key advantage is the enhanced connectivity, offering potentially shorter routes for parallel packet transmissions and thereby reducing communication latency. This solution presents superior flexibility compared to conventional 2D-NoCs. Furthermore, we proposed a WCTT analysis based on DAG for multi-dimensional NoCs, and an upper-bound on the packets' WCIT is also derived.

All NoCs were implemented in a hardware description language.

8.2 Thesis Validation

In Section 1.3, we established the following thesis statement:

It is possible to design and implement a predictable deflection-based NoC architecture for real-time systems that not only provides different levels of QoS and offers deterministic and easy-to-analyze timing behaviors but also guarantees in-order flit delivery at the destination and ensures efficient network bandwidth utilization, while remaining suitable for implementation in both low-end and high-end chips, and for which a comprehensive analysis of the worst-case communication time of packets can be provided.

In this dissertation, we propose and implement a set of NoC architectures: HopliteRT*, IPDeN, IPDeN 2.0, and nDimNoC, each exploring various design choices. These solutions rely on the concept of deflection to minimize the requirement for extensive buffers and VCs in the communication infrastructure of upcoming MPSoCs. Some of the proposed NoC architectures, specifically IPDeN and IPDeN 2.0, incorporate buffers into their design. However, the buffer size depends solely on the network topology and is independent of the set of communication flows traveling through the network. This design approach guarantees predictable performance and ensures that the silicon and power footprint of the NoC remains small and independent of the specific applications running on the network. Additionally, we provide a formally proven correct WCCT analysis for each of these NoCs. The traversal time analyses derived in Sections 3.4.1, 4.4, 5.3, and 7.4.1, only utilize information from the flit under analysis and do not rely on information from other flows traversing the network. This characteristic makes these analyses particularly valuable for applications with dynamically varying workloads and timing properties, offering predictable performance without the need for over-provisioning or dynamic network reconfiguration.

Under the Orion test case, IPDeN outperforms 128-deep HopliteBuf for WMTT and AMTT in approximately 66% of flows, while HopliteBuf demonstrates better performance in around 20% of flows, with 14% of flows exhibiting similar traversal times for both NoCs. Nonetheless, a significant issue arises in HopliteBuf, where 79583 out of 301746 flits (26% of the total) fail to reach their destination due to router buffer overflow. IPDeN also demonstrates superior performance compared to the VCs-based NoC for WMTT, WMCT, and AMCT in approximately 43-56% of flows. The VCs-based NoC outperforms IPDeN in AMTT for 67% of flows, but this performance improvement requires a significant increase in hardware resources. On the other hand, IPDeN 2.0 overcomes the performance of 128-deep HopliteBuf in terms of WMTT, WMCT, AMTT, and AMCT in approximately 55-60% of flows. Conversely, HopliteBuf shows better performance in roughly 17-21% of flows, with 19-28% of flows exhibiting similar results for both NoCs. However, in HopliteBuf, 1653 out of 202039 flits never reach their destination. When comparing IPDeN 2.0 to the VCs-based NoC, around 77% of flows experience reduced WMCT with IPDeN 2.0, while the remaining 23% see reduced WMCT in the VCs-based NoC. IPDeN 2.0 also significantly outperforms the VCs-based NoC in AMCT for about 90% of flows, with only 20 flows exhibiting AMCT degradation with IPDeN 2.0. When evaluating different configurations of nDimNoC with a 16x16 HopliteRT* under random traffic patterns, the average WCTT is reduced by approximately 40% with a 5D-nDimNoC and approximately 60% with a 6D-nDimNoC compared to HopliteRT*. Packets also experience a significant reduction in WCIT when using nDimNoC compared to HopliteRT*. The average WCCT improves as the network dimensionality

increases in nDimNoC and is minimally affected by the number of communication flows in the network.

In Section 3.4.2, we derived more precise upper-bounds on the WCTT of packets traversing HopliteRT* by considering the set of communication flows that interfere with the transmission of the flit under analysis. In scenarios involving random traffic patterns, we observed a significant improvement in the WCTT and average WCTT of high-priority packets when they traverse HopliteRT*, and the improved analysis is used, compared to its predecessor, HopliteRT. Specifically, under a 16x16 NoC configuration, the WCTT and average WCTT of high-priority packets are reduced by half while maintaining the same quality of service for low-priority ones. Furthermore, when only a few flows are traveling through the networks, the average WCTT of high-priority packets is enhanced by up to 5x.

While injection times may vary depending on the workload, in Chapter 6, we introduce a novel solution for regulating traffic injection in NoCs. This solution effectively limits the maximum bandwidth that can be claimed by each PE and/or communication flow. In the absence of the D-TSR protocol, in HopliteRT*, IPDeN, IPDeN 2.0, and nDimNoC, multiple PEs can only start injecting flits once other PEs have no more flits to inject into the network. However, when our D-TSR protocol is integrated into any of these NoC architectures, the allocation of network bandwidth by PEs is regulated by the parameters G_q and D_q . Additionally, in Section 6.4, we derive an injection time analysis that is agnostic and well-suited for dynamic applications in which the set of communication flows can change over time.

In Sections 3.5.1, 4.6.1, 5.4.1, and 7.5.1, we reported that an 8x8 HopliteRT*, an 8x8 IPDeN, an 8x8 IPDeN 2.0, and a 4x4x4 3D-nDimNoC NoCs require approximately 15-times, 3-times, 2-times, and 5-times fewer hardware resources, respectively, than some commonly used real-time NoC solutions based on wormhole switching with VCs. The NoC architectures proposed in this dissertation are well-suited for systems with power and/or area constraints and limited resource availability, such as those implemented on mid-range FPGA platforms. After allocating hardware resources on these platforms, our NoCs still leave resources available for computational logic.

In conclusion, our proposed NoC solutions guarantee predictable timing behaviors, adapt to dynamic workloads, and maintain a low hardware footprint, aligning perfectly with common industrial communication requirements. Furthermore, we have developed a practical timing analysis tool that has been formally proven to be correct, thereby validating the thesis statement.

8.3 Future work

In this section, we identify potential directions for extending our research in the future.

8.3.1 Fault Tolerance

Given the increasing complexity and criticality of applications, managing faults in communication links, routers, and other NoC components is crucial. Fault tolerance mechanisms are essential to prevent disruptions, maintain predictable communication latencies, and ensure smooth operation

in the presence of component failures or disruptions. In this context, exploring fault tolerance for real-time deflection-based NoCs is an interesting research line. That is, developing techniques to handle link or router failures while ensuring predictable communication latencies. This research has the potential to strengthen the resilience of deflection-based NoCs, thus advancing communication infrastructure in modern SoCs.

8.3.2 Energy Efficiency

In this research, our focus was on evaluating the hardware resource utilization of the proposed NoC architectures. However, an interesting direction for future exploration would be to delve into techniques aimed at optimizing energy consumption in deflection-based NoCs. This could involve optimizing data transmission strategies, minimizing the utilization of deflection and buffering mechanisms, and/or establishing low-power modes to achieve more energy-efficient communication.

8.3.3 Applications and Workload Scenarios

It would be valuable to investigate the applicability of deflection-based NoCs in specific domains, such as multimedia processing, automotive systems, or edge computing. This involves identifying optimal configurations for diverse workload scenarios within these domains.

Appendix A

Counter-Example to the WCIT Bound of [90, 91]

The bound for the worst-case injection time of packets transmitted over HopliteRT derived by Wasly et al. in [90, 91] is incorrect as it may return optimistic and hence unsafe results. We use a counter-example to explain the flaw. The approach is then as follows.

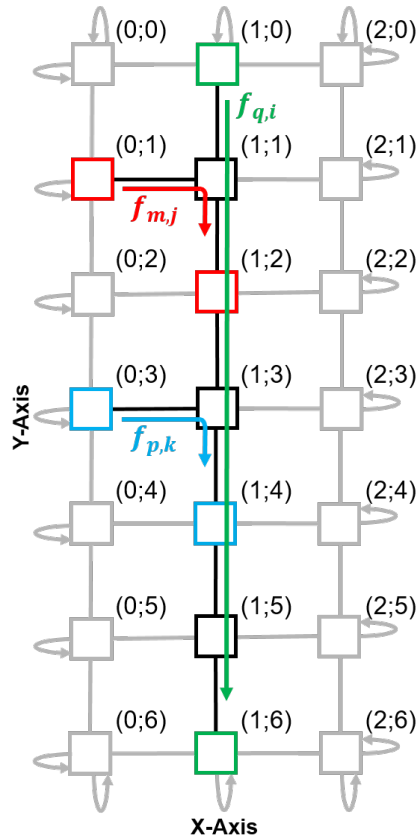


Figure A.1: Counter-example to the WCIT bound of [90, 91]

Note that, Equation 2.3 assumes that a conflicting flow inherits the burst length and injection period of its origin router, i.e., $\sigma_{m,j}$ and $\theta_{m,j}$ refer to the burst length and regulation period of the origin router of flow $f_{m,j}$. Then, consider the system presented in Figure A.1. It consists of three flows $f_{q,i}$, $f_{m,j}$ and $f_{p,k}$. $f_{q,i}$ is injected into router at coordinates (1;0) and has destination router at coordinates (1,6). $f_{m,j}$ is injected into router at coordinates (0,1) and has destination router at coordinates (1,2). $f_{p,k}$ is injected into router at coordinates (0,3) and has destination router at coordinates (1,4).

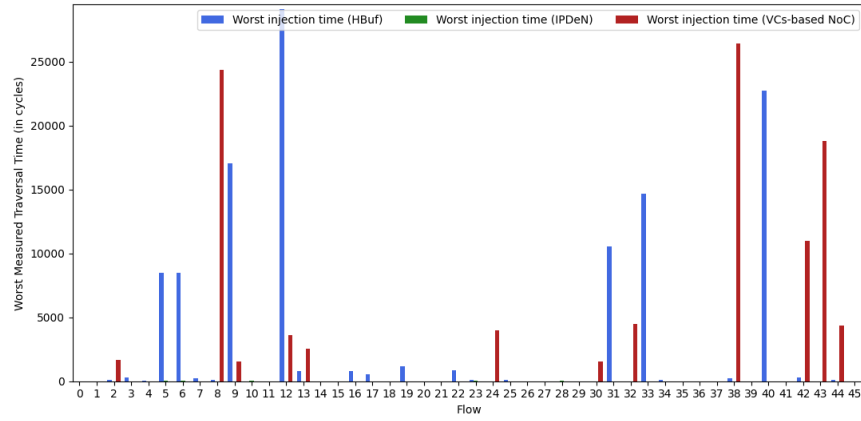
We are interested about the maximum blocking delay B_l suffered by a flit of a flow $f_{s,l}$ injected to the port S of router at coordinate (1,5). The set of conflicting flows $\Gamma_{s,l}^C = \{f_{q,i}\}$, i.e., only $f_{q,i}$ pass through the port S of router (1,5) and block flit of a flow $f_{s,l}$. Assume now that the burst length $\sigma_{q,i} = 1$ and the regulation period $\theta_{q,i} = 4$. Then, according to Equation 2.3, the maximum number of clock cycles port S of router (1,5) may be kept busy by conflicting flows is given by $B_l = \left\lceil \frac{\sigma(\Gamma_{s,l}^C)}{1-\rho(\Gamma_{s,l}^C)} \right\rceil = \left\lceil \frac{1}{1-\frac{1}{4}} \right\rceil = 2$.

However, let's now suppose that $f_{q,i}$ injects three flits at times 0, 4, and 8, respecting thus the regulation period of 4 at the origin router of $f_{q,i}$. Let us additionally assume that $f_{m,j}$ injects flits at times 0 and 4. Then, the first two flits of $f_{q,i}$ are deflected by flits of $f_{m,j}$ in router (1,1), while the third flit of $f_{q,i}$ passes freely through port S of (1,1). Therefore, the three flits of $f_{q,i}$ reach router (1,2) at times 5, 9, and 10, respectively. Now suppose that flow $f_{p,k}$ injects a flit at time 5. Then, the first flit of $f_{q,i}$ is deflected a second time in router (1,3). That is, the three flits of $f_{q,i}$ reach router (1,5) at time 11, 12, and 13, respectively. Since they all arrive in one clock cycle of the interval, this means that port S of router (1,5) is kept busy by the conflicting flow $f_{q,i}$ during 3 clock cycles, which contradicts Equation 2.3.

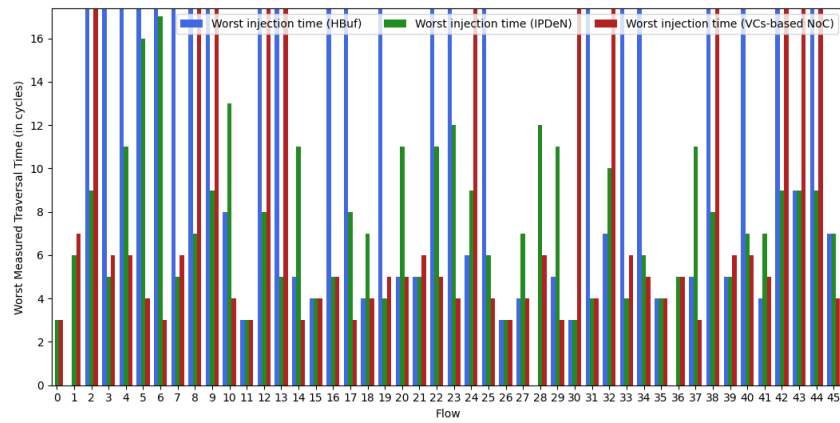
We conclude then that the analysis is incorrect as it forgets to take into account the fact that different flits of the same flow may suffer a different number of deflections on their route to their destination.

Appendix B

Experimental Results for IPDeN NoC

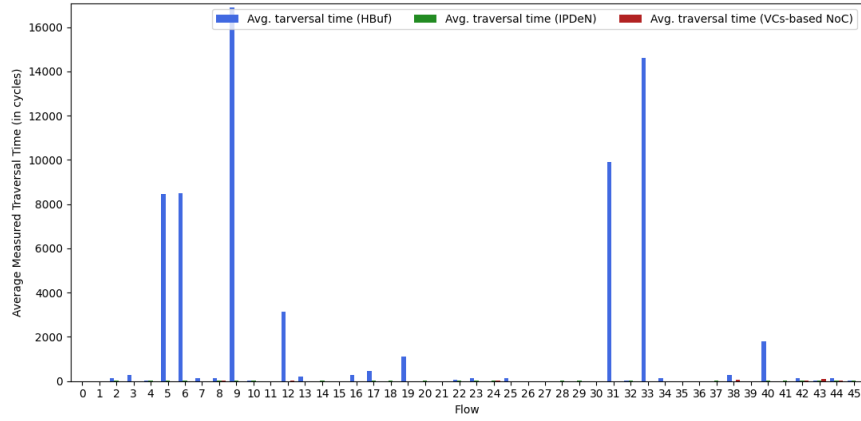


(a) WMTT.

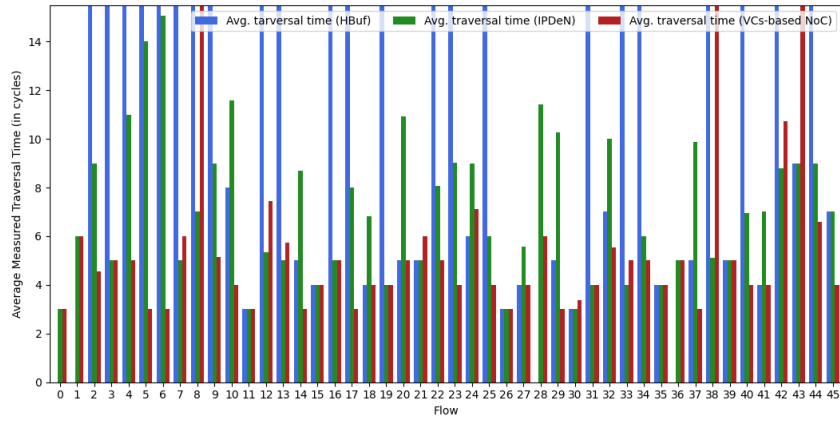


(b) WMTT (Zoom).

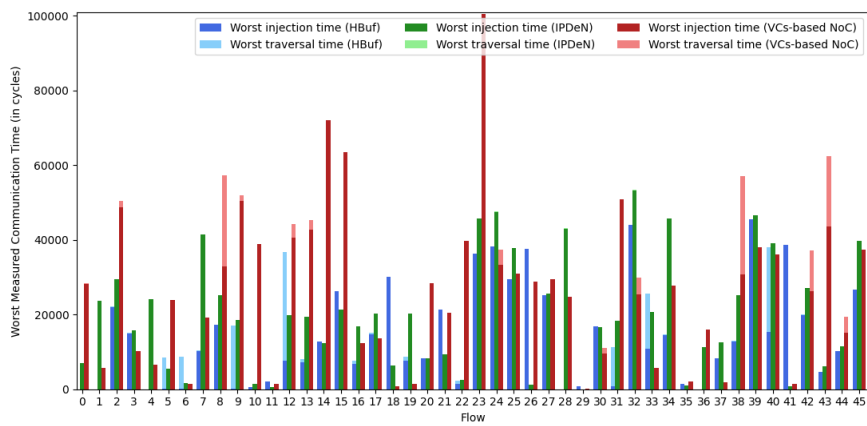
Figure B.1: Experimental results for Synthetic test case.



(c) AMTT.

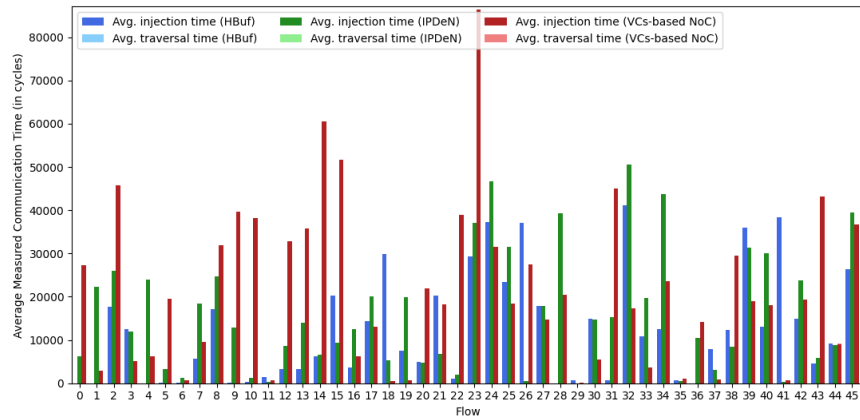


(d) AMTT (Zoom).



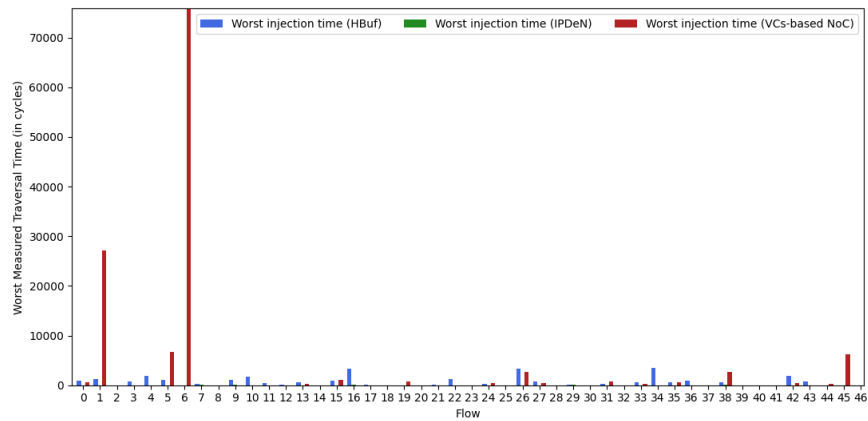
(e) WMCT.

Figure B.1: Experimental results for Synthetic test case (cont. 1).

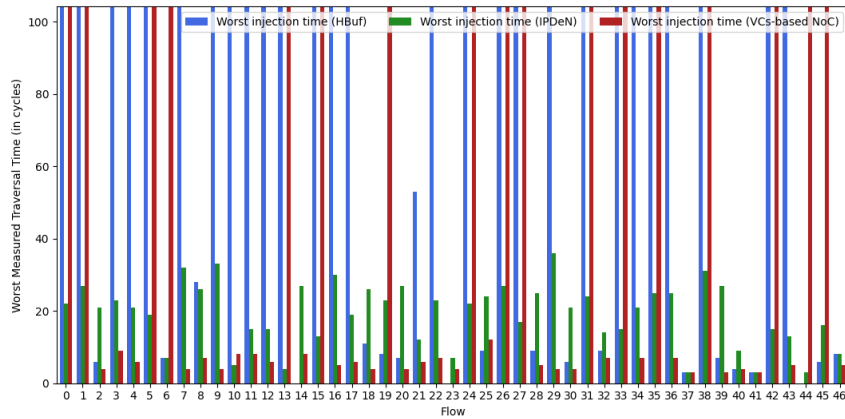


(f) AMCT.

Figure B.1: Experimental results for Synthetic test case (cont. 2).

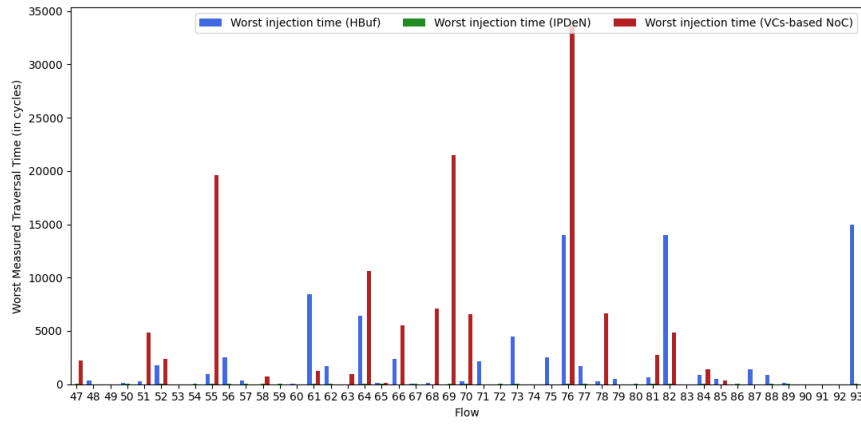


(a) WMTT 1-5.

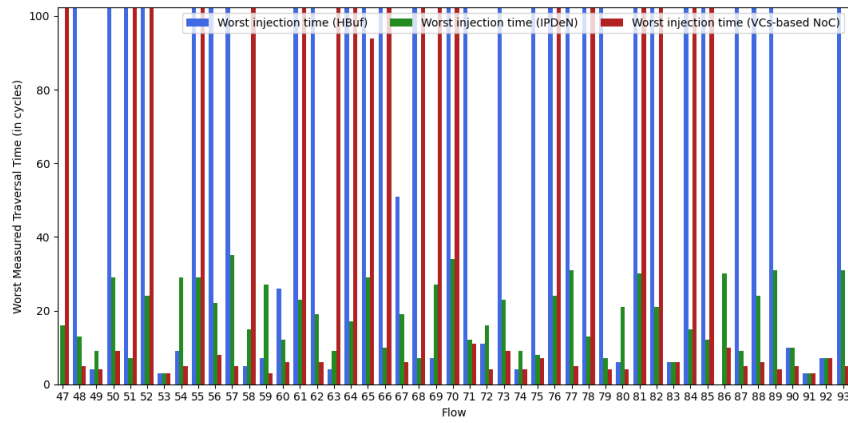


(b) WMTT 1-5 (Zoom).

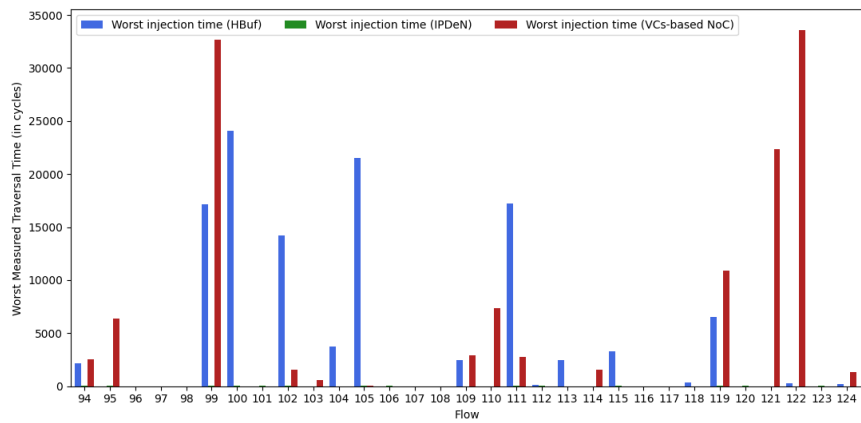
Figure B.2: Experimental results for Orion test case (WMTT).



(c) WMTT 2-5.

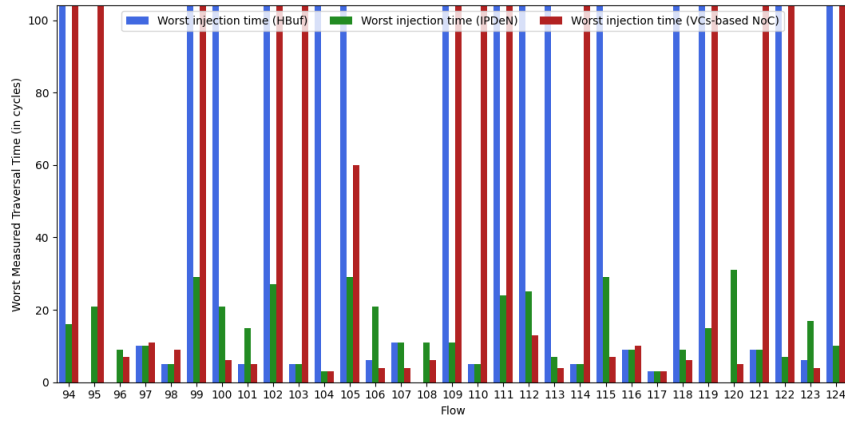


(d) WMTT 2-5 (Zoom).

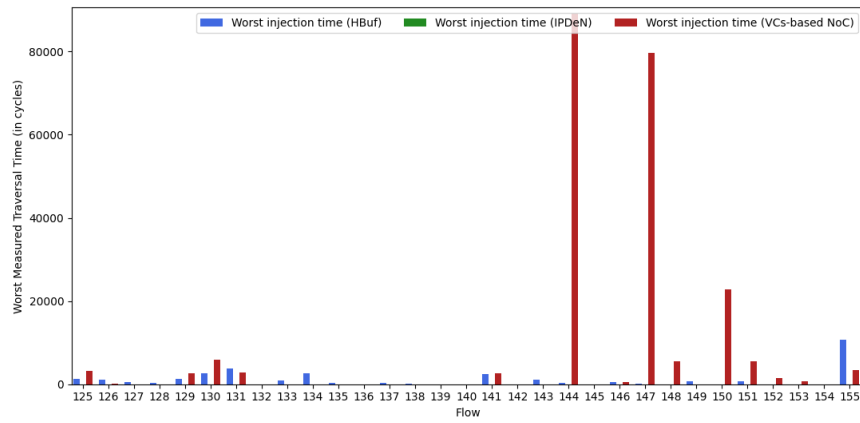


(e) WMTT 3-5.

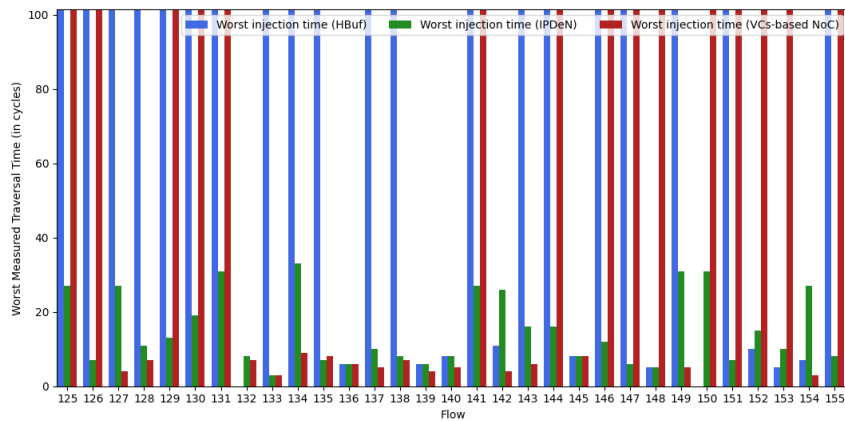
Figure B.2: Experimental results for Orion test case (WMTT) (cont. 1).



(f) WMTT 3-5 (Zoom).

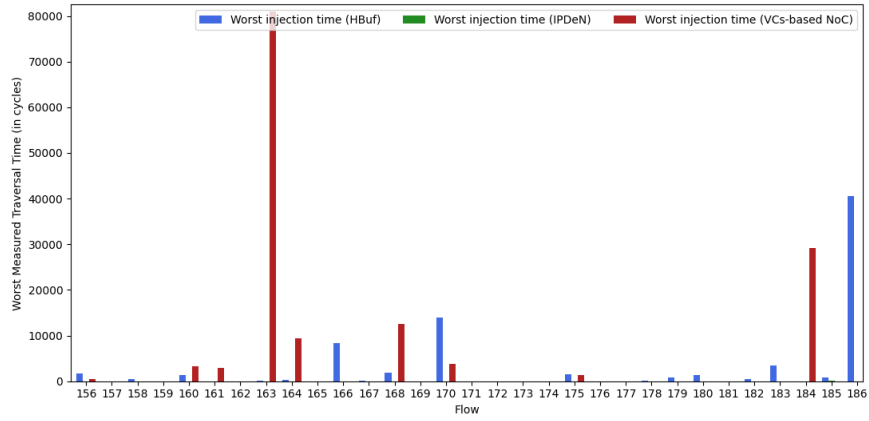


(g) WMTT 4-5.

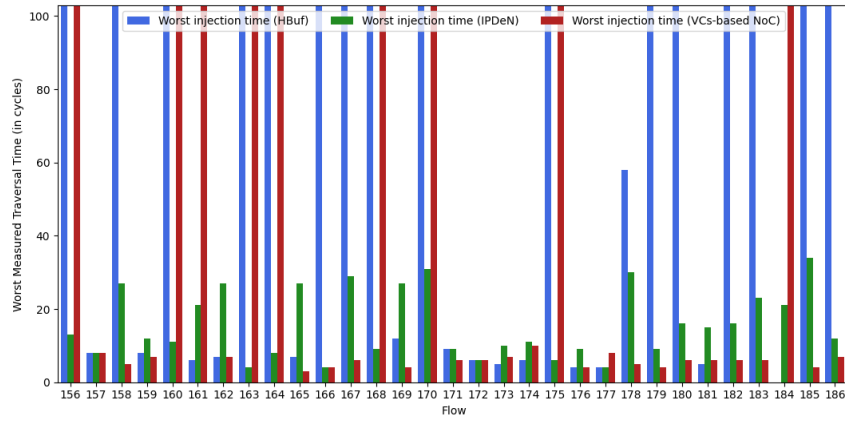


(h) WMTT 4-5 (Zoom).

Figure B.2: Experimental results for Orion test case (WMTT) (cont. 2).

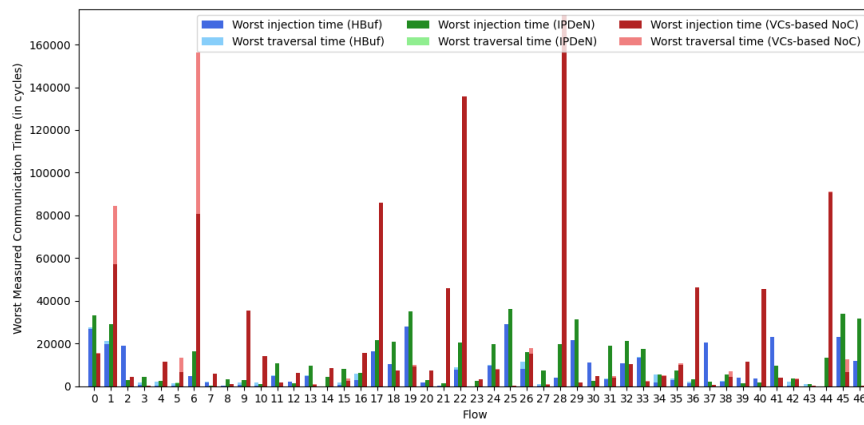


(i) WMTT 5-5.



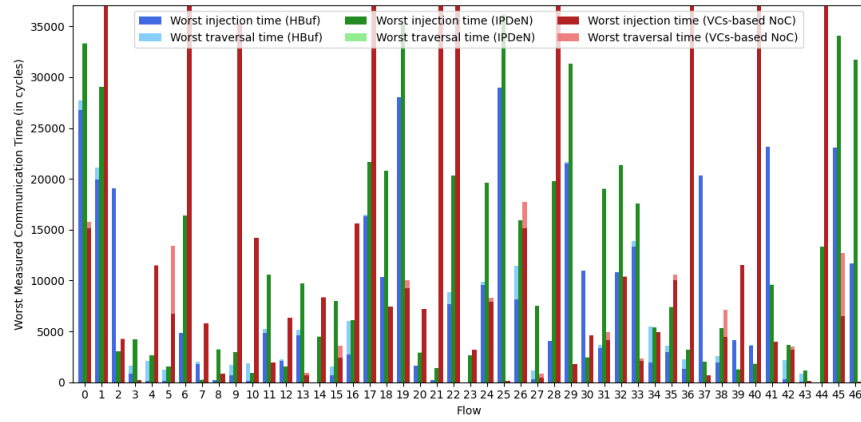
(j) WMTT 5-5 (Zoom).

Figure B.2: Experimental results for Orion test case (WMTT) (cont. 3).

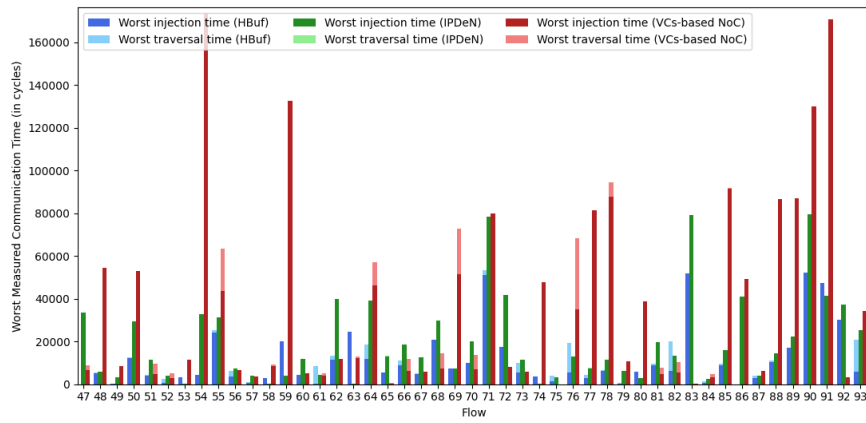


(a) WMCT 1-5.

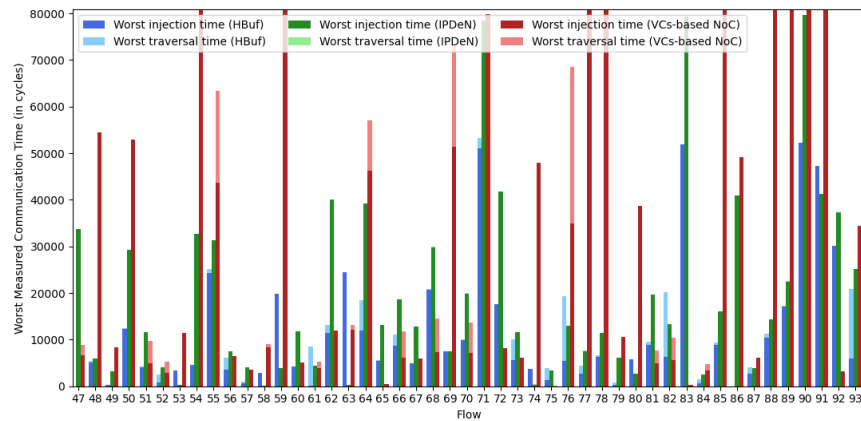
Figure B.3: Experimental results for Orion test case (WMCT).



(b) WMCT 1-5 (Zoom).

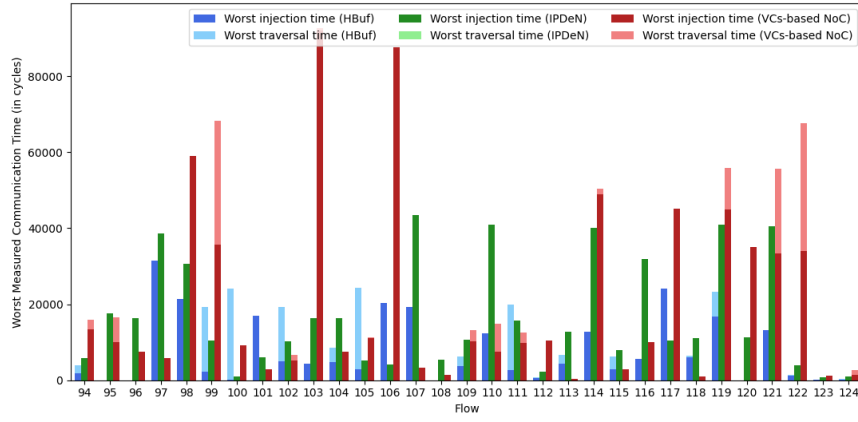


(c) WMCT 2-5.

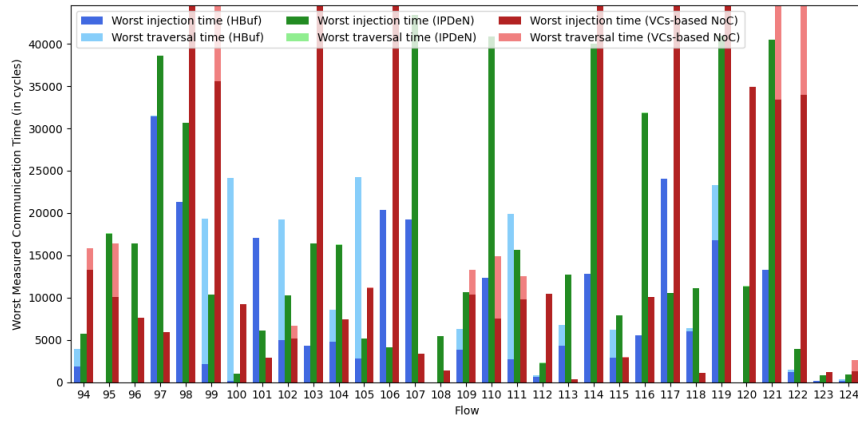


(d) WMCT 2-5 (Zoom).

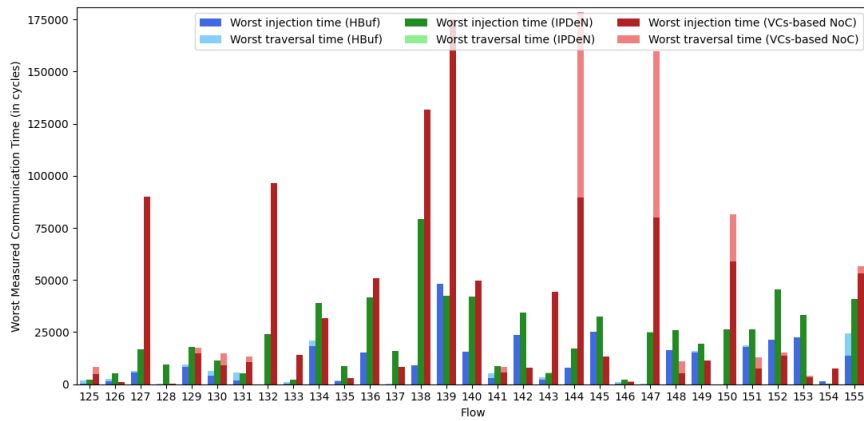
Figure B.3: Experimental results for Orion test case (WMCT) (cont. 1).



(e) WMCT 3-5.

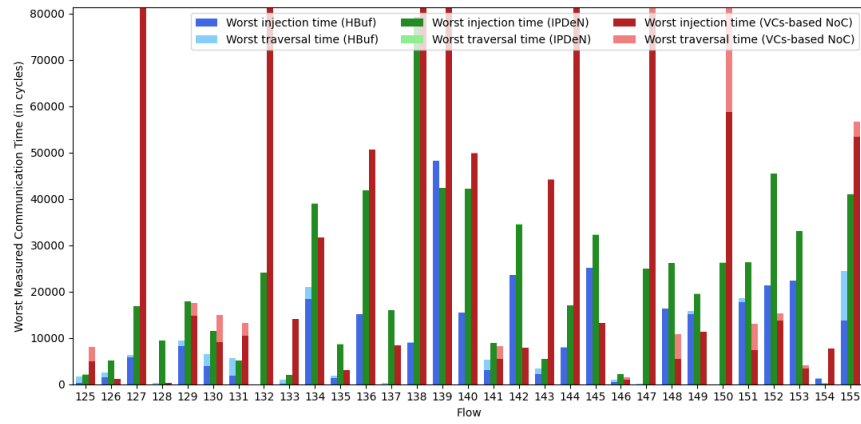


(f) WMCT 3-5 (Zoom).

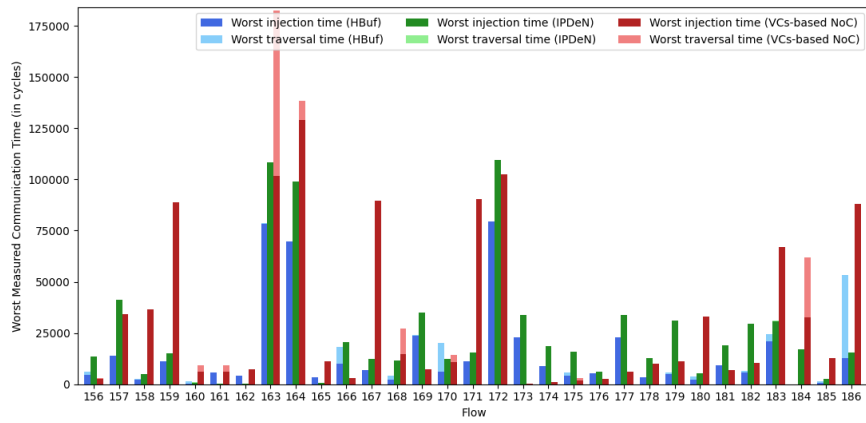


(g) WMCT 4-5.

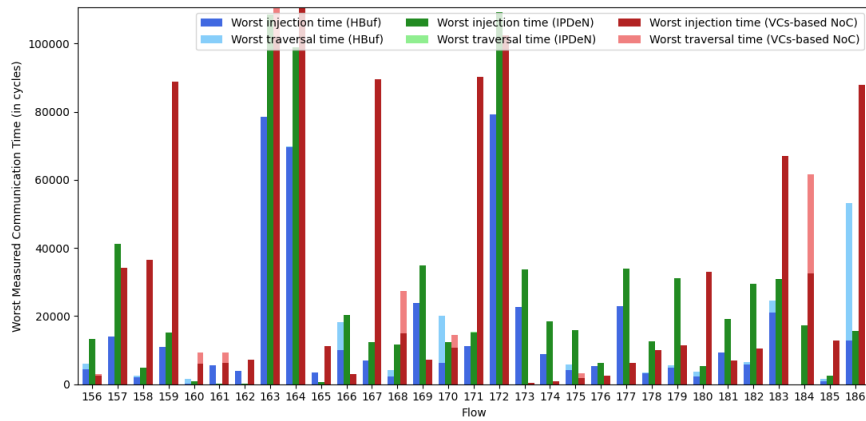
Figure B.3: Experimental results for Orion test case (WMCT) (cont. 2).



(h) WMCT 4-5 (Zoom).

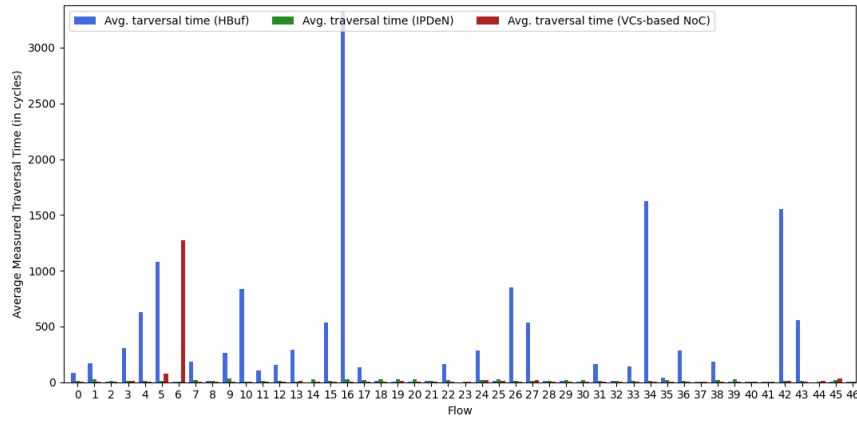


(i) WMCT 5-5.

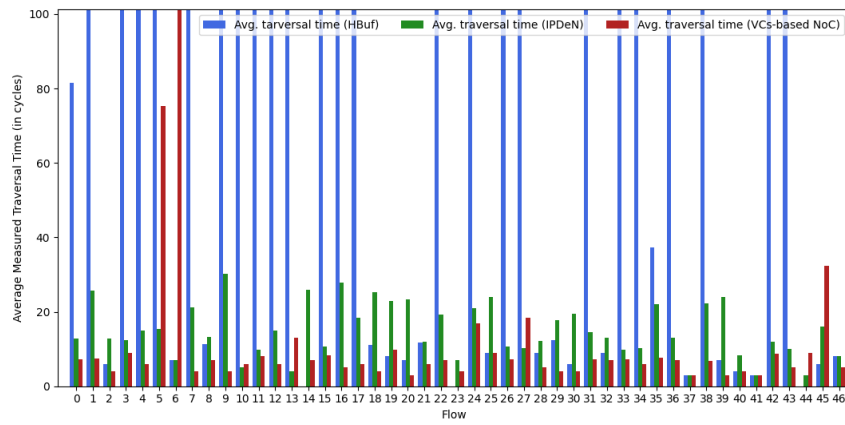


(j) WMCT 5-5 (Zoom).

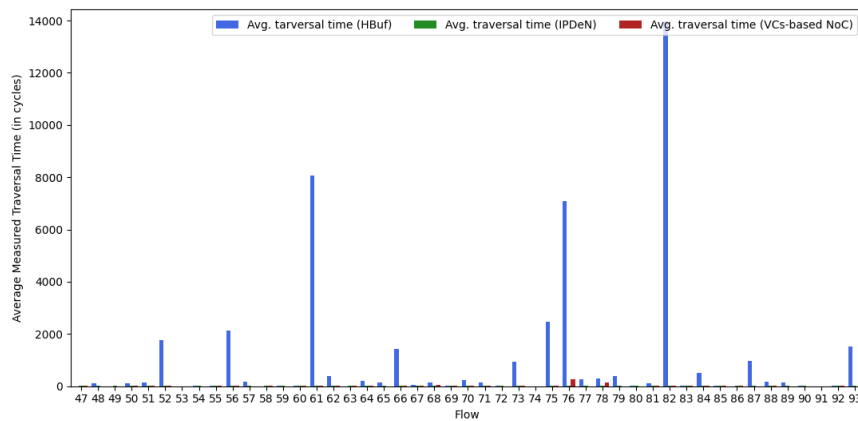
Figure B.3: Experimental results for Orion test case (WMCT) (cont. 3).



(a) AMTT 1-5.

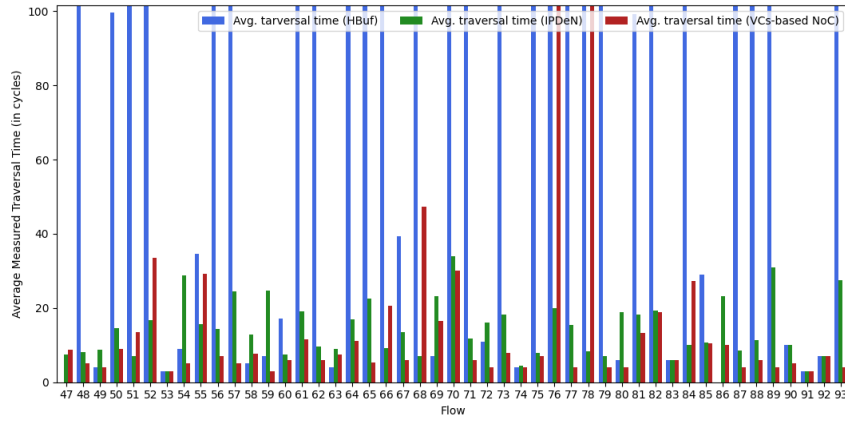


(b) AMTT 1-5 (Zoom).

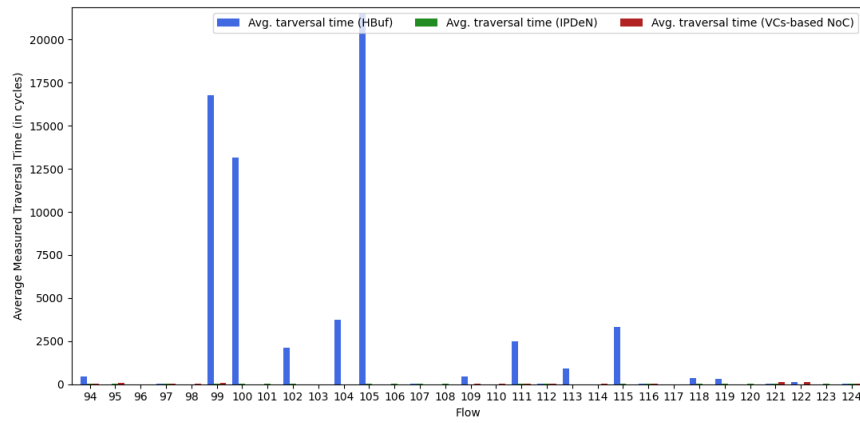


(c) AMTT 2-5.

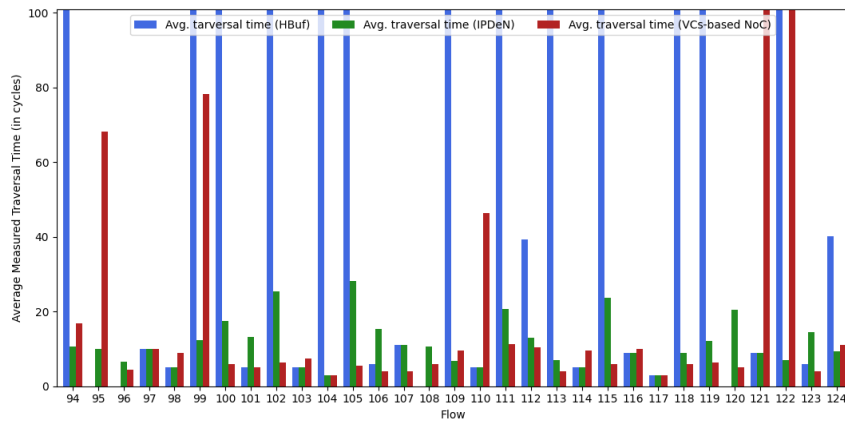
Figure B.4: Experimental results for Orion test case (AMTT).



(d) AMTT 2-5 (Zoom).

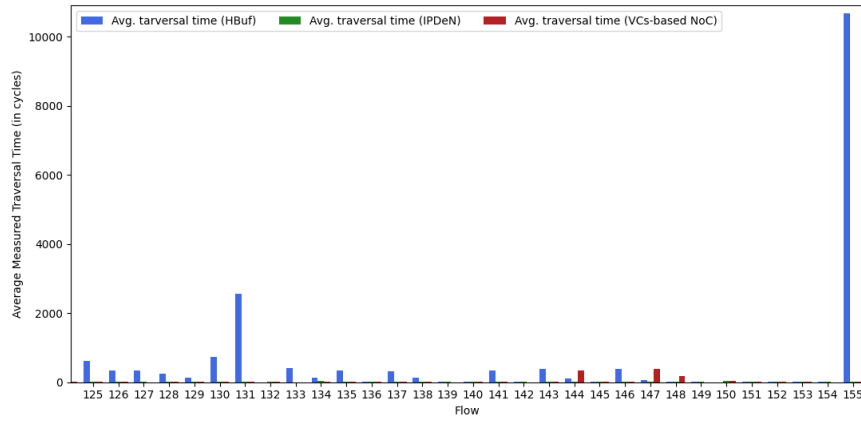


(e) AMTT 3-5.

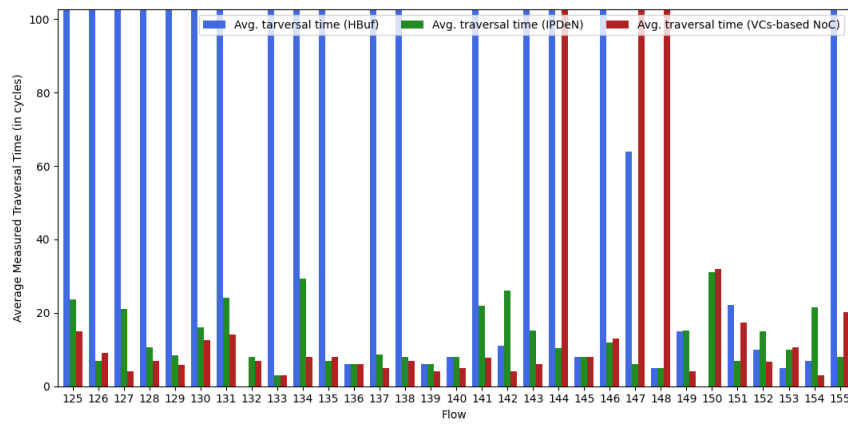


(f) AMTT 3-5 (Zoom).

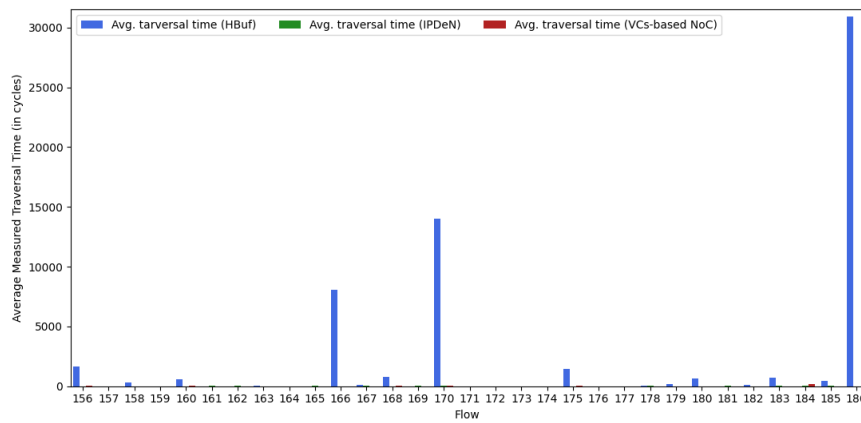
Figure B.4: Experimental results for Orion test case (AMTT) (cont.1).



(g) AMTT 4-5.

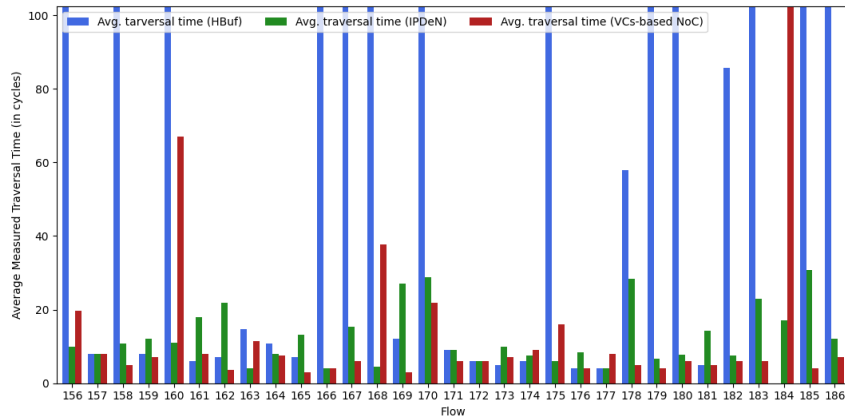


(h) AMTT 4-5 (Zoom).



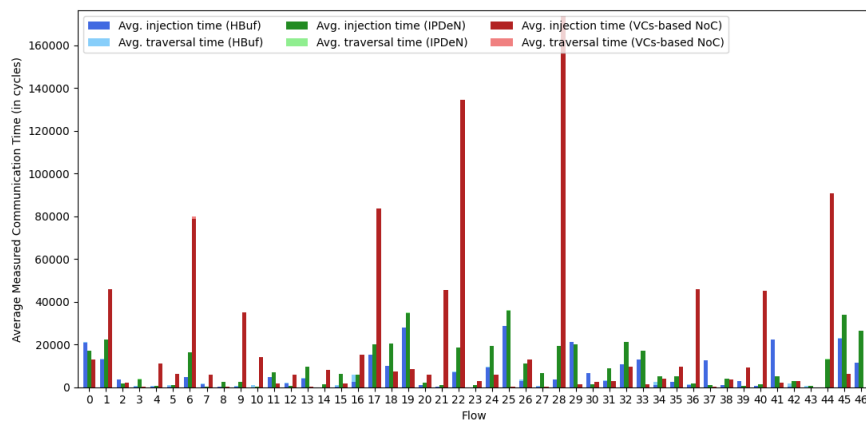
(i) AMTT 5-5.

Figure B.4: Experimental results for Orion test case (AMTT) (cont.2).

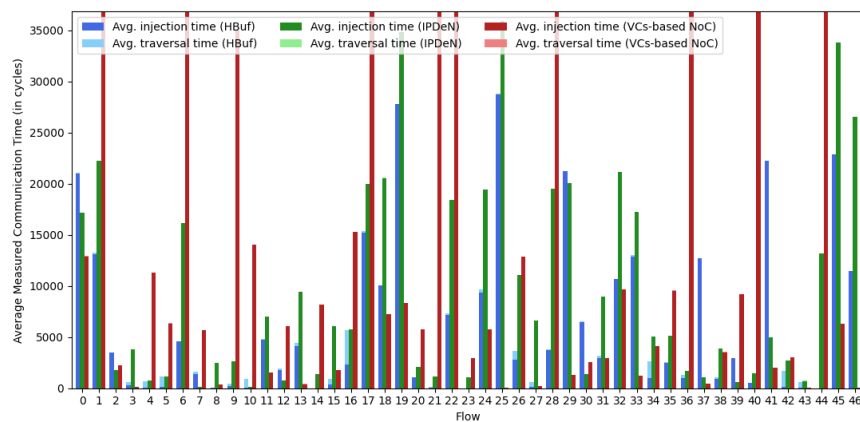


(j) AMTT 5-5 (Zoom).

Figure B.4: Experimental results for Orion test case (AMTT) (cont. 3).

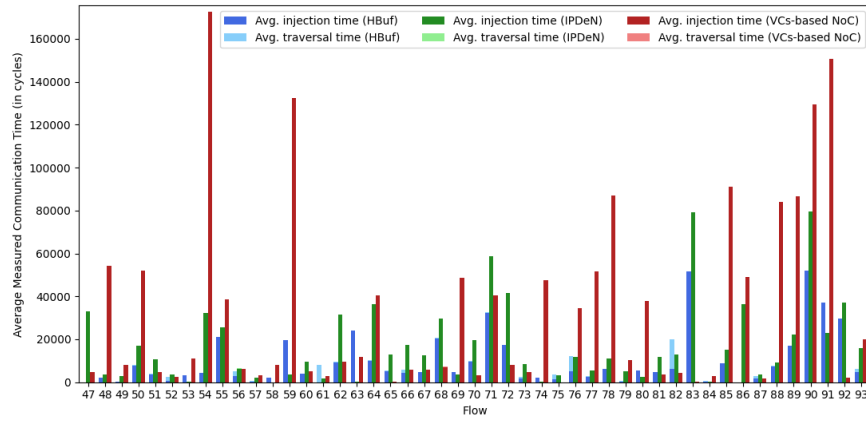


(a) AMCT 1-5.

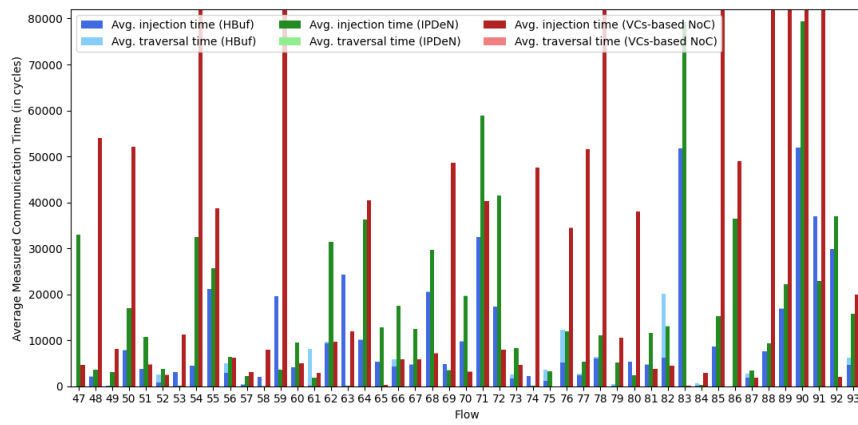


(b) AMCT 1-5 (Zoom).

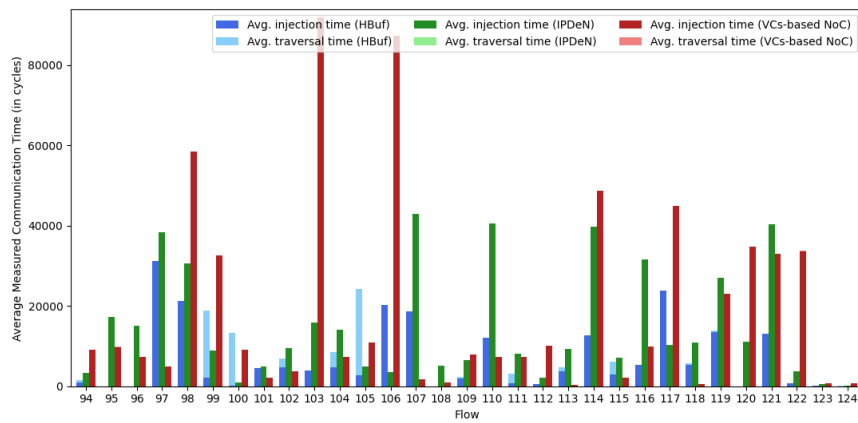
Figure B.5: Experimental results for Orion test case (AMCT).



(c) AMCT 2-5.

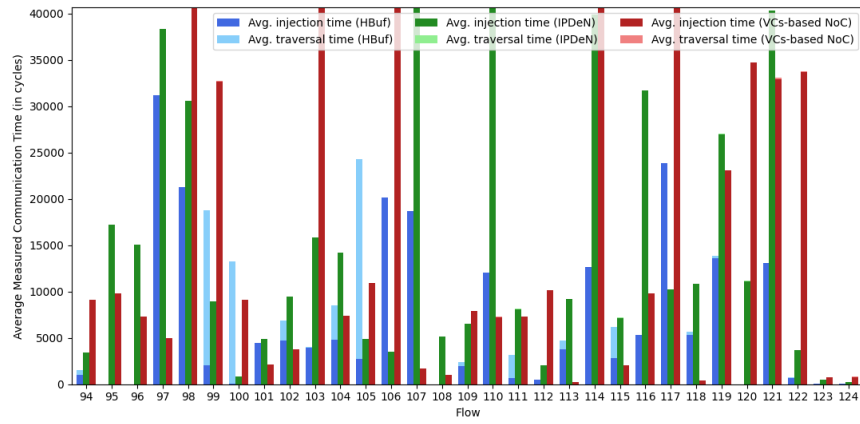


(d) AMCT 2-5 (Zoom).

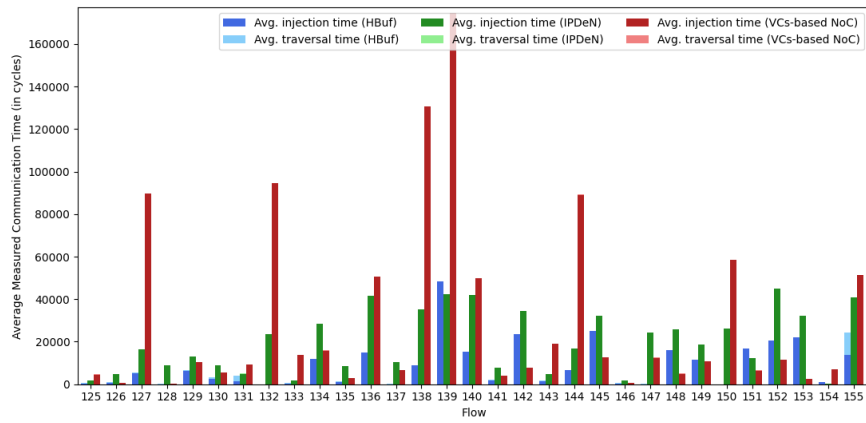


(e) AMCT 3-5.

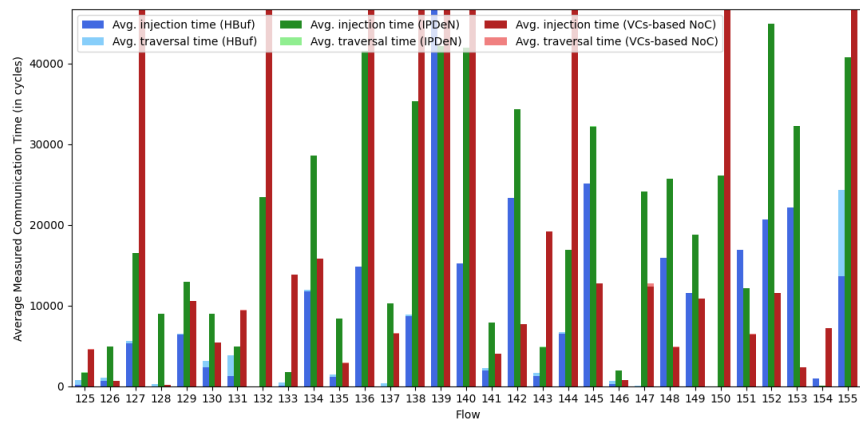
Figure B.5: Experimental results for Orion test case (AMCT) (cont. 1).



(f) AMCT 3-5 (Zoom).

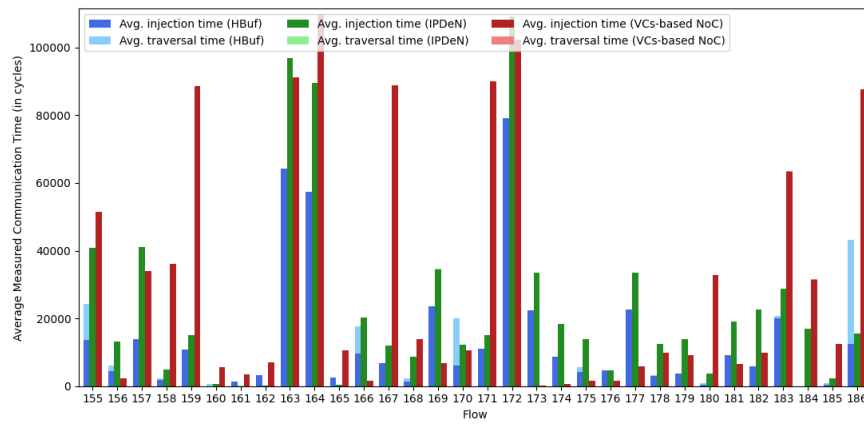


(g) AMCT 4-5.



(h) AMCT 4-5 (Zoom).

Figure B.5: Experimental results for Orion test case (AMCT) (cont. 2).

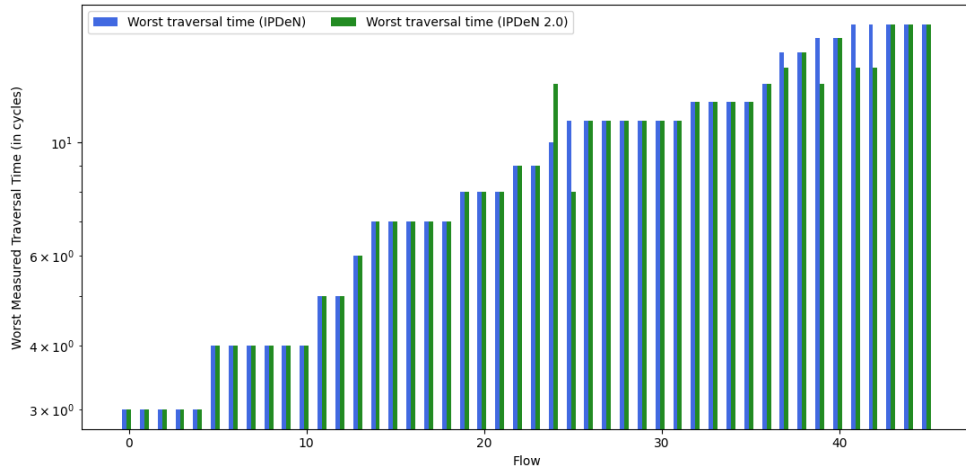


(i) AMCT 5-5.

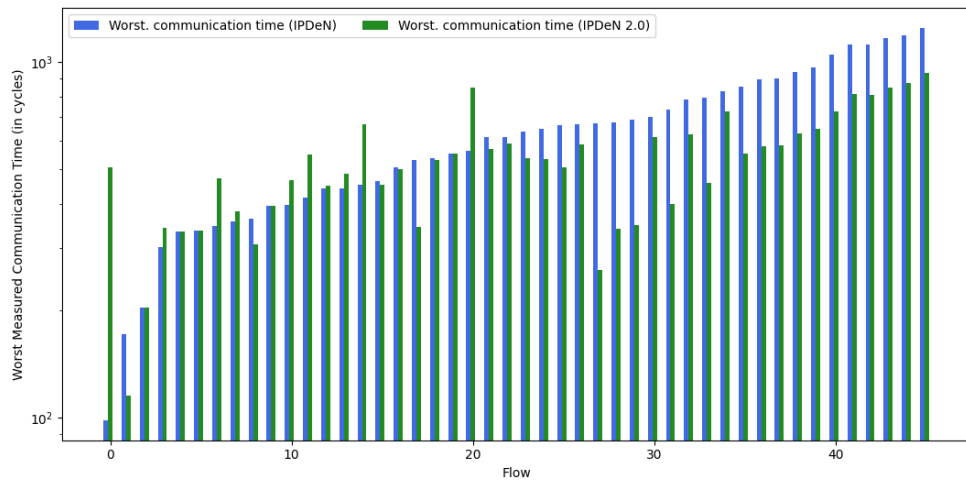
Figure B.5: Experimental results for Orion test case (AMCT) (cont. 3).

Appendix C

Experimental Results for IPDeN 2.0 NoC

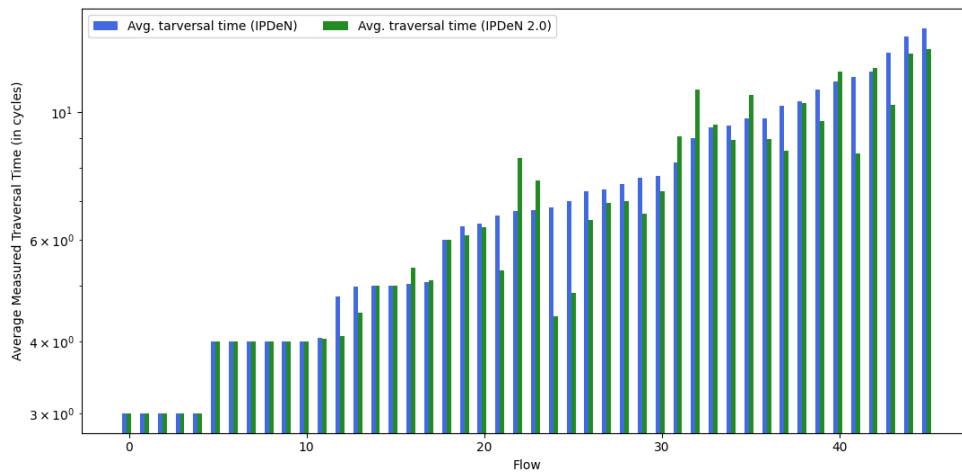


(a) Worst measured traversal time.

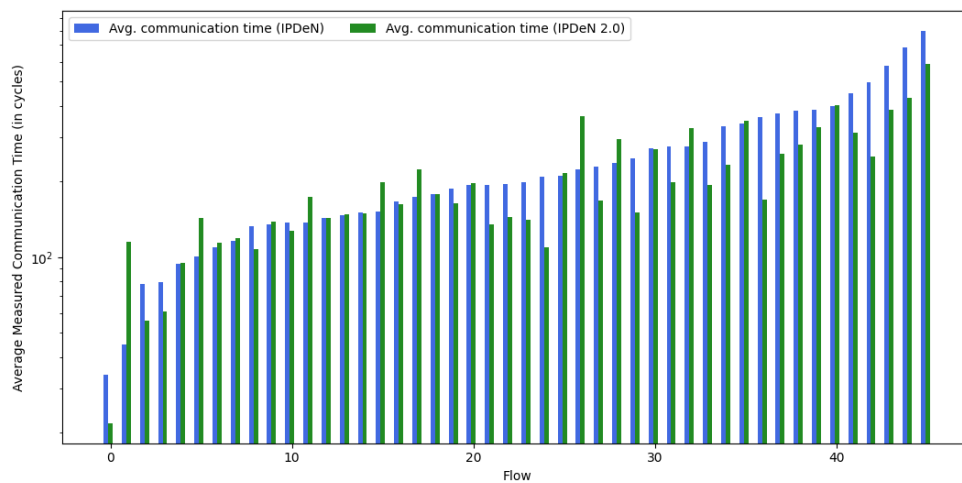


(b) Worst measured communication time.

Figure C.1: IPDeN 2.0 vs IPDeN (Synthetic test case).

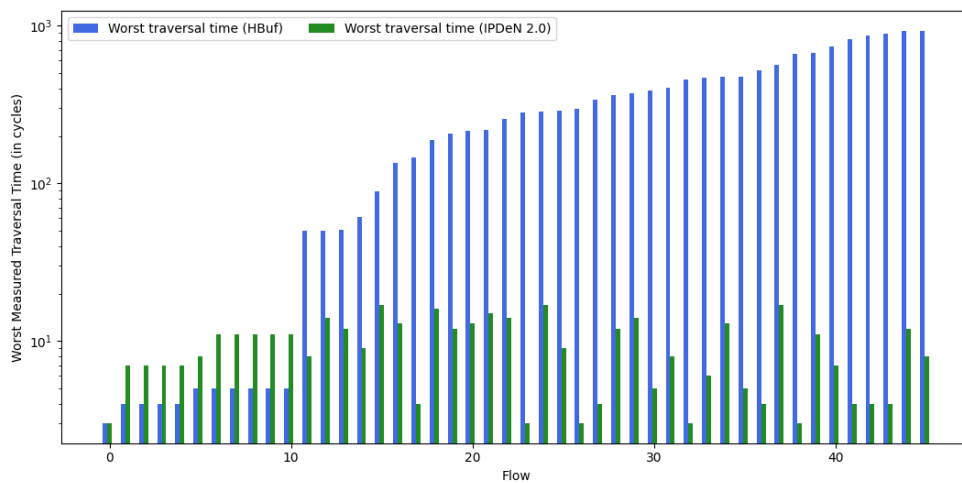


(c) Average measured traversal time.



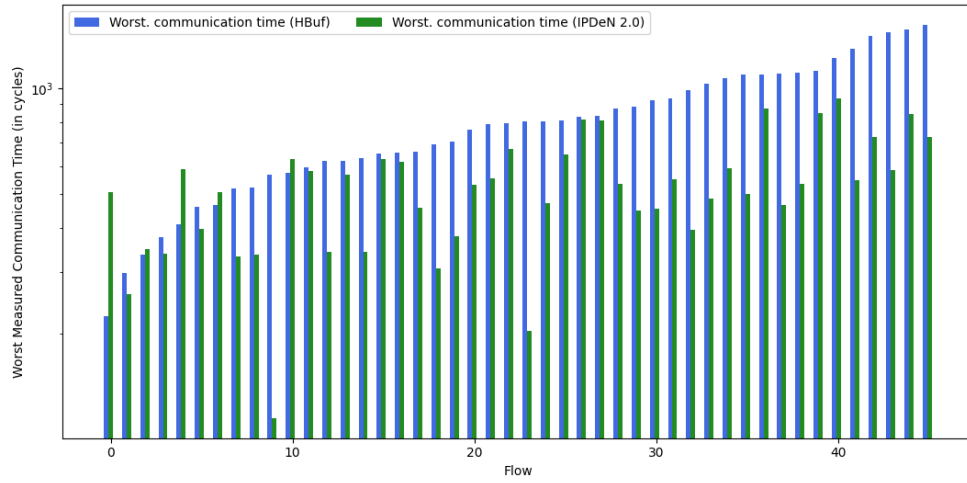
(d) Average measured communication time.

Figure C.1: IPDeN 2.0 vs IPDeN (Synthetic test case) (cont.).

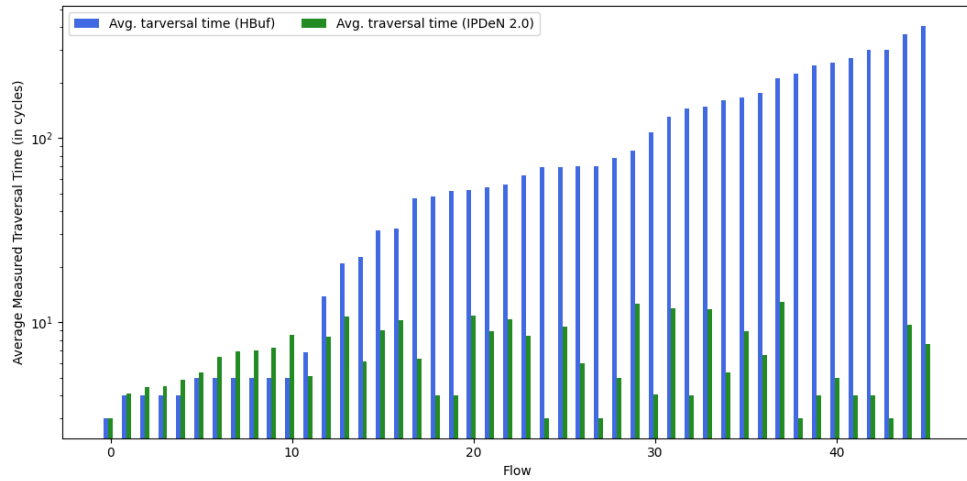


(a) Worst measured traversal time.

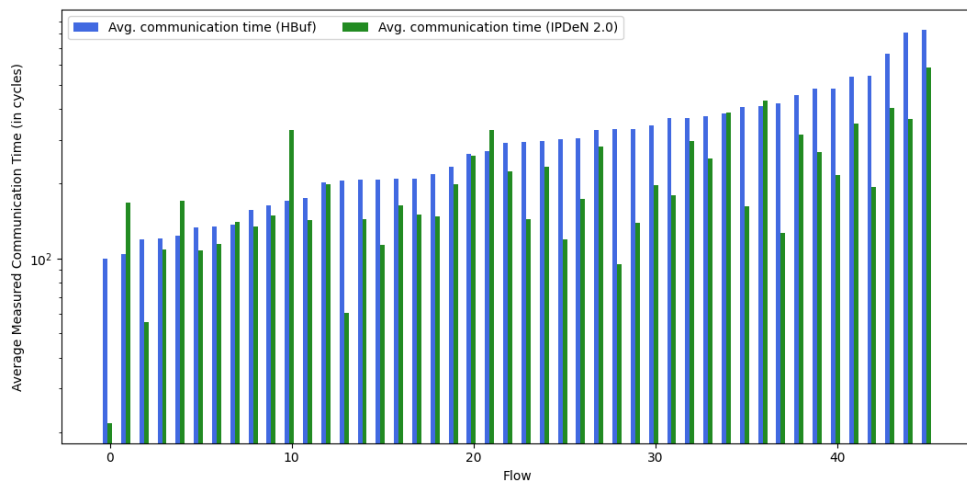
Figure C.2: IPDeN 2.0 vs 128-deep HopliteBuf (Synthetic test case).



(b) Worst measured communication time.

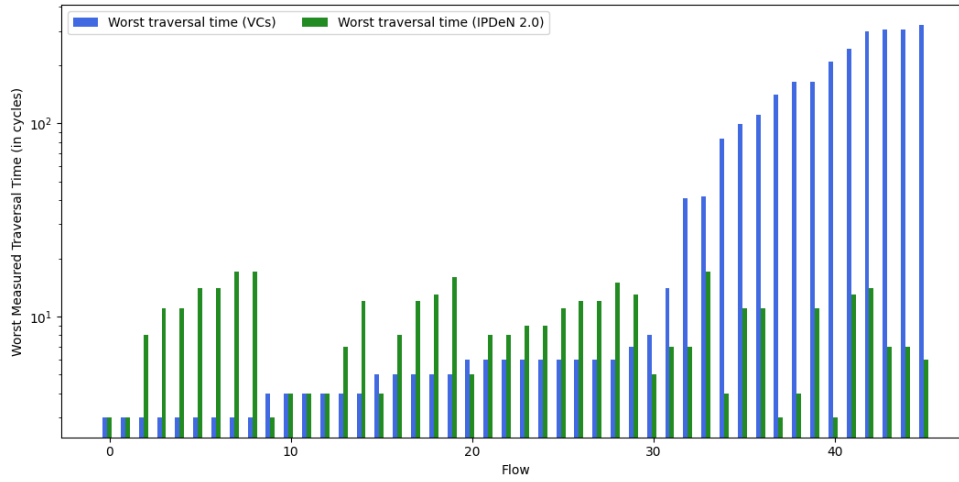


(c) Average measured traversal time.

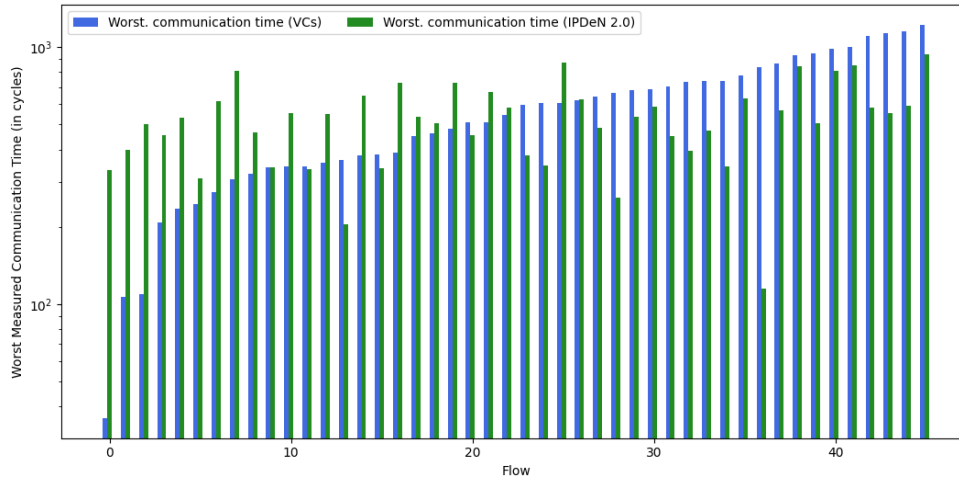


(d) Average measured communication time.

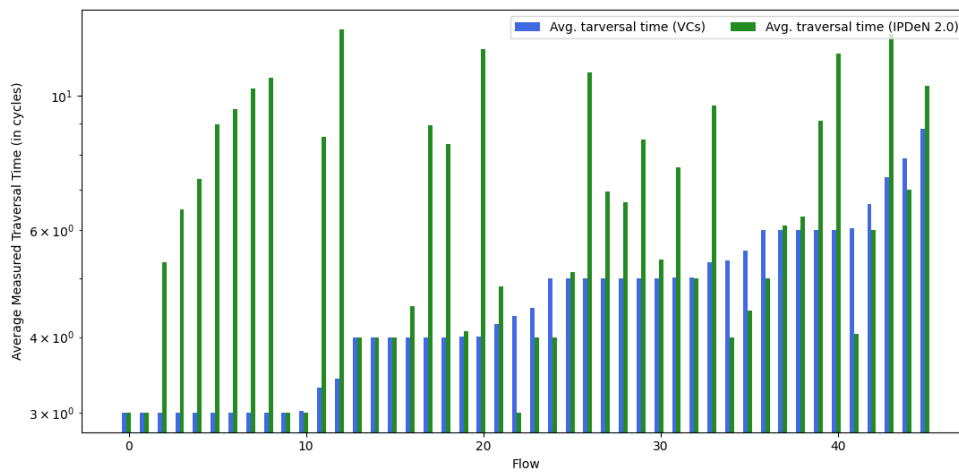
Figure C.2: IPDeN 2.0 vs 128-deep HopliteBuf (Synthetic test case) (cont.).



(a) Worst measured traversal time.

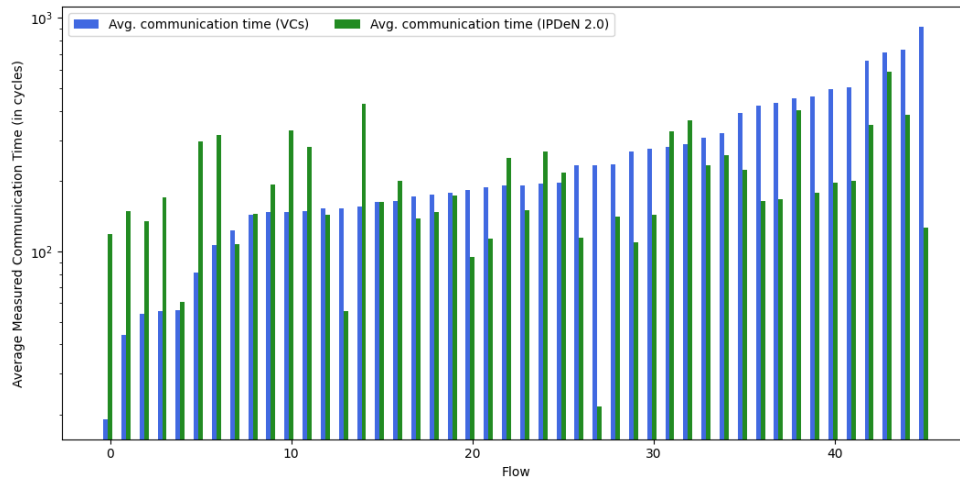


(b) Worst measured communication time.



(c) Average measured traversal time.

Figure C.3: IPDeN 2.0 vs VCs (Synthetic test case) (cont.).



(d) Average measured communication time.

Figure C.3: IPDeN 2.0 vs VCs (Synthetic test case) (cont.).

Table C.1: Gain on the WMTT (Orion test case).

% of gain	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
0%	126	53	29
(0%,10%]	19	2	1
(10%,20%]	15	0	12
(20%,30%]	12	2	2
(30%,40%]	11	1	4
(40%,50%]	4	1	4
(50%,60%]	0	4	3
(60%,70%]	0	2	4
(70%,80%]	0	10	2
(80%,90%]	0	20	8
(90%,100%]	0	60	32

Table C.2: Gain on the WMCT (Orion test case).

% of gain	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
0%	85	41	0
(0%,10%]	59	14	8
(10%,20%]	14	8	7
(20%,30%]	10	14	10
(30%,40%]	7	17	18
(40%,50%]	3	14	46
(50%,60%]	1	22	15
(60%,70%]	0	14	10
(70%,80%]	0	3	21
(80%,90%]	0	3	7
(90%,100%]	0	1	1

Table C.3: Gain on the AMTT (Orion test case).

% of gain	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
0%	74	51	28
(0%,10%]	47	8	20
(11%,20%]	36	7	12
(20%,30%]	16	2	4
(30%,40%]	6	6	7
(40%,50%]	2	4	4
(50%,60%]	1	7	0
(60%,70%]	0	13	3
(70%,80%]	0	12	0
(80%,90%]	0	24	1
(90%,100%]	0	20	0

Table C.4: Gain on the AMCT (Orion test case).

% of gain	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
0%	46	35	0
(0%,10%]	106	31	8
(10%,20%]	18	23	13
(20%,30%]	7	20	18
(30%,40%]	0	7	26
(40%,50%]	0	12	67
(50%,60%]	0	11	16
(60%,70%]	0	7	11
(70%,80%]	0	0	4
(80%,90%]	0	1	4
(90%,100%]	0	1	0

Table C.5: Lost on the WMTT (Orion test case).

% of lost	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
(0%,10%]	0	0	0
(10%,20%]	0	2	6
(20%,30%]	0	1	4
(30%,40%]	0	2	3
(40%,50%]	0	2	8
(50%,60%]	0	1	3
(60%,70%]	0	0	0
(70%,80%]	0	3	6
(80%,90%]	0	3	5
(90%,100%]	0	3	11
(100%,inf]	0	15	40

Table C.6: Lost on the WMCT (Orion test case).

% of lost	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
(0%,10%]	5	19	7
(10%,20%]	1	4	6
(20%,30%]	1	3	5
(30%,40%]	0	5	0
(40%,50%]	0	2	2
(50%,60%]	0	0	0
(60%,70%]	1	0	4
(70%,80%]	0	0	4
(80%,90%]	0	0	1
(90%,100%]	0	0	1
(100%,inf]	0	3	14

Table C.7: Lost on the AMTT (Orion test case).

% of lost	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
(0%,10%]	4	9	12
(10%,20%]	0	10	20
(20%,30%]	0	4	7
(30%,40%]	0	2	4
(40%,50%]	0	2	12
(50%,60%]	0	0	7
(60%,70%]	0	3	5
(70%,80%]	0	0	6
(80%,90%]	0	0	1
(90%,100%]	0	0	7
(100%,inf]	0	3	29

Table C.8: Lost on the AMCT (Orion test case).

% of lost	Number of flows		
	IPDeN 2.0 vs IPDeN	IPDeN 2.0 vs 128-deep HopliteBuf	IPDeN 2.0 vs VC
(0%,10%]	7	20	1
(10%,20%]	3	10	2
(20%,30%]	0	3	1
(30%,40%]	0	2	6
(40%,50%]	0	0	0
(50%,60%]	0	1	2
(60%,70%]	0	1	1
(70%,80%]	0	1	2
(80%,90%]	0	0	0
(90%,100%]	0	0	1
(100%,inf]	0	1	4

References

- [1] A. Agarwal and R. Shankar. Survey of network on chip (NoC) architectures & contributions. *Journal of Engineering, Computing and Architecture*, 3, jan 2009.
- [2] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. An empirical survey-based study into industry practice in real-time systems. In *2020 IEEE Real-Time Systems Symposium*, pages 3–11, 2020.
- [3] Miguel Gorgues Alonso, José Flich, Meriem Turki, and Davide Bertozzi. A low-latency and flexible TDM NoC for strong isolation in security-critical systems. In *IEEE 13th Int. Symp. on Emb. Multi/Many-core SoC*, pages 149–156, 2019.
- [4] Shobana Balakrishnan and Fusun Ozguner. A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):664–678, 1998.
- [5] P. Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, 1964.
- [6] Matthias Becker, Borislav Nikolic, Dakshina Dasari, Benny Akesson, Vincent Nélis, Moris Behnam, and Thomas Nolte. Partitioning and analysis of the network-on-chip on a cots many-core platform. In *Real-Time and Embedded Technology and Applications Symposium*, pages 101–112. IEEE, 2017.
- [7] L. Benini and G. De Micheli. Networks on chip: a new paradigm for systems on chip design. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 418–419, March 2002.
- [8] Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30:129–154, 2005.
- [9] T. Bjerregaard and J. Sparso. Implementation of guaranteed services in the MANGO clockless network-on-chip. *IEE Proceedings - Computers and Digital Techniques*, 153(4):217–229, July 2006.
- [10] Tobias Bjerregaard and Jens Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *Design, Automation and Test in Europe*, pages 1226–1231. IEEE, 2005.
- [11] Rahul Boyapati, Jiayi Huang, Pritam Majumder, Ki Hwan Yum, and Eun Jung Kim. Approx-noc: A data approximation framework for network-on-chip architectures. In *ISCA*, pages 666–677, 2017.

- [12] A. Burns and Robert I. Davis. Mixed criticality systems - a review. Technical report, University of York, 2019.
- [13] Alan Burns, James Harbin, and Leandro Soares Indrusiak. A wormhole NoC protocol for mixed criticality systems. In *IEEE Real-Time Systems Symposium*, pages 184–195, 2014.
- [14] J. Chen, P. G. Gillard, and C. Y. Li. Network-on-chip (NoC) topologies and performance: A review. 2011.
- [15] Yiou Chen, Jianhao Hu, Xiang Ling, and Tingting Huang. A novel 3D NoC architecture based on De Bruijn graph. *Computers & Electrical Engineering*, 38(3):801–810, 2012.
- [16] Rene L Cruz. A calculus for network delay. I. Network elements in isolation. *IEEE Transactions on information theory*, 37(1):114–131, 1991.
- [17] William J. Dally. Virtual-channel flow control. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ISCA '90, pages 60–68, New York, NY, USA, 1990. Association for Computing Machinery.
- [18] Dakshina Dasari, Borislav Nikolić, Vincent N’elis, and Stefan M Petters. NoC contention analysis using a branch-and-prune algorithm. *ACM Transactions on Embedded Computing Systems*, 13(3s):113, 2014.
- [19] Benoît Dupont de Dinechin, Yves Durand, Duco van Amstel, and Alexandre Ghiti. Guaranteed services of the NoC of a manycore processor. In *NoCArc*, pages 11–16, 2014.
- [20] Benoît Dupont De Dinechin, Duco Van Amstel, Marc Poulhiès, and Guillaume Lager. Time-critical computing on a single-chip massively parallel processor. In *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2014.
- [21] Berend HJ Dekens, Philip S Wilmanns, Gerard JM Smit, and Marco JG Bekooij. Low-cost guaranteed-throughput dual-ring communication infrastructure for heterogeneous mpsoes. In *Proceedings of the 2014 Conference on Design and Architectures for Signal and Image Processing*, pages 1–8. IEEE, 2014.
- [22] J. Diemer and R. Ernst. Back suction: Service guarantees for latency-sensitive on-chip networks. In *Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 155–162, May 2010.
- [23] Jonas Diemer, Jonas Rox, Mircea Negrean, Steffen Stein, and Rolf Ernst. Real-time communication analysis for networks with two-stage arbitration. In *9th ACM International Conference on Embedded Software*. IEEE, 2011.
- [24] Yves Durand, Christian Bernard, and Fabien Clermidy. Distributed dynamic rate adaptation on a network on chip with traffic distortion. In *MCSOC*, pages 225–232. IEEE, 2016.
- [25] Ian Elmor Lang, Rodolfo Pellizzoni, and Nachiket Kapre. HopliteRT source queuing bound correction. Technical report, 2020.
- [26] Feihui Li, C. Nicopoulos, T. Richardson, Yuan Xie, V. Narayanan, and M. Kandemir. Design and management of 3d chip multiprocessors using network-in-memory. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 130–141, 2006.

- [27] Thomas Ferrandiz, Fabrice Frances, and Christian Fraboul. A method of computation for worst-case delay analysis on spacewire networks. In *2009 IEEE International Symposium on Industrial Embedded Systems*, pages 19–27. IEEE, 2009.
- [28] T. Garg, S. Wasly, R. Pellizzoni, and N. Kapre. HopliteBuf: FPGA NoCs with provably stall-free FIFOs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '19*, pages 222–231, New York, NY, USA, 2019. ACM.
- [29] Yan Ghidini, Thais Webber, Edson Moreno, Fernando Grando, Rubem Fagundes, and César Marcon. Buffer depth and traffic influence on 3d nocs performance. In *2012 23rd IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 9–15. IEEE, 2012.
- [30] Frédéric Giroudot and Ahlem Mifdaoui. Buffer-aware worst-case timing analysis of wormhole NoCs using network calculus. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2018.
- [31] Frederic Giroudot and Ahlem Mifdaoui. Tightness and computation assessment of worst-case delay bounds in wormhole networks-on-chip. In *27th International Conference on Real-Time Networks and Systems*, 2019.
- [32] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *19th Annual International Symposium on Computer Architecture*, pages 278–287, May 1992.
- [33] Yilian Ribot González, Geoffrey Nelissen, and Eduardo Tovar. IPDeN: Real-time deflection-based NoC with in-order flits delivery. In *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 160–169. IEEE, 2022.
- [34] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers*, 22(5):414–421, Sep. 2005.
- [35] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh. A scalable communication-centric soc interconnect architecture. In *International Symposium on Signals, Circuits and Systems. Proceedings, SCS 2003. (Cat. No.03EX720)*, pages 343–348, 2004.
- [36] R. I. Greenberg and Lee Guan. An improved analytical model for wormhole routed networks with application to butterfly fat-trees. In *Proceedings of the 1997 International Conference on Parallel Processing (Cat. No.97TB100162)*, pages 44–48, 1997.
- [37] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, pages 250–256, 2000.
- [38] Jörg Henkel, Wayne Wolf, and Srimat Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. In *17th International Conference on VLSI Design*. IEEE, 2004.
- [39] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany. Survey on real-time networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1500–1517, May 2017.
- [40] D. M. Holman and D. C. S. Lee. A survey of routing techniques in store-and-forward and wormhole interconnects. 2008.

- [41] L. S. Indrusiak, A. Burns, and B. Nikolić. Buffer-aware bounds to multi-point progressive blocking in priority-preemptive NoCs. In *2018 Design, Automation Test in Europe Conference Exhibition*, 2018.
- [42] Leandro Soares Indrusiak, James Harbin, and Alan Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *27th Euromicro Conference on Real-Time Systems*. IEEE, 2015.
- [43] Intel. Single-chip-cloud computer. www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf, 2010.
- [44] Intel. Intel @xeon phiTM. www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html, 2013.
- [45] Fahimeh Jafari, Zhonghai Lu, Axel Jantsch, and Mohammad H Yaghmaee. Optimal regulation of traffic flows in networks-on-chip. In *Design, Automation Test in Europe Conference Exhibition*, pages 1621–1624. IEEE, 2010.
- [46] Fahimeh Jafari, Zhonghai Lu, Axel Jantsch, and Mohammad Hossein Yaghmaee. Buffer optimization in network-on-chip through flow regulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):1973–1986, 2010.
- [47] Fahimeh Jafari, Mohammad H Yaghmaee, Mohammad S Talebi, and Ahmad Khonsari. Max-min-fair best effort flow control in network-on-chip architectures. In *International Conference on Computational Science*, pages 436–445. Springer, 2008.
- [48] Kalray. Mppa-256 manycore processor. www.kalrayinc.com/kalray/products/processors, 2014.
- [49] N. Kapre and J. Gray. Hoplite: Building austere overlay NoCs for FPGAs. In *25th International Conference on Field Programmable Logic and Applications*, pages 1–8, Sep 2015.
- [50] Hany Kashif, Sina Gholamian, and Hiren Patel. Sla: A stage-level latency analysis for real-time communication in a pipelined resource model. *IEEE Transactions on Computers*, 64(4):1177–1190, 2014.
- [51] Byungjae Kim, Jong Kim, Sungje Hong, and Sunggu Lee. A real-time communication method for wormhole switching networks. In *Proceedings. 1998 International Conference on Parallel Processing (Cat. No. 98EX205)*, pages 527–534. IEEE, 1998.
- [52] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.
- [53] Meng Liu, Matthias Becker, Moris Behnam, and Thomas Nolte. Tighter time analysis for real-time traffic in on-chip networks with shared priorities. In *10th IEEE/ACM International Symposium on Networks-on-Chip*, 2016.
- [54] Zhonghai Lu, Axel Jantsch, and Ingo Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 960–964, 2005.
- [55] Zhonghai Lu, Mikael Millberg, Axel Jantsch, Alistair Bruce, Pieter van der Wolf, and Tomas Henriksson. Flow regulation for on-chip communication. In *Design, Automation Test in Europe Conference Exhibition*, pages 578–581. IEEE, 2009.

- [56] Zhonghai Lu and Yuan Yao. Dynamic traffic regulation in noc-based systems. *VLSI*, 25(2):556–569, 2016.
- [57] César Marcon, Ramon Fernandes, Rodrigo Cataldo, Fernando Grando, Thais Webber, Ana Benso, and Letícia B Poehls. Tiny noc: A 3d mesh topology with router channel optimization for area and latency minimization. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 228–233. IEEE, 2014.
- [58] Jörg Mische and Theo Ungerer. Low power flitwise routing in an unidirectional torus with minimal buffering. In *Proceedings of the Fifth International Workshop on Network on Chip Architectures*, pages 63–68, 2012.
- [59] Alireza Monemi, Jia Tang, Maurizio Palesi, and Muhammad Nadzir Marsono. ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform. *Microprocessors and Microsystems*, 54, 09 2017.
- [60] Georgios-Grigorios Mplemenos and Ioannis Papaefstathiou. Mplem: An 80-processor FPGA based multiprocessor system. In *16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2008.
- [61] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb 1993.
- [62] B. Nikolić and S. M. Petters. Edf as an arbitration policy for wormhole-switched priority-preemptive NoCs-Myth or fact? In *International Conference on Embedded Software*, pages 1–10, Oct 2014.
- [63] B. Nikolić, S. Tobuschat, L. Soares, R. Ernst, and A. Burns. Real-time analysis of priority-preemptive NoCs with arbitrary buffer sizes and router delays. *Real-Time Systems*, 55(1):63–105, Jan 2019.
- [64] Ioannis Nouisias and Tughrul Arslan. Wormhole routing with virtual channels using adaptive rate control for network-on-chip (NoC). In *AHS*, pages 420–423. IEEE, 2006.
- [65] Umit Y Ogras and Radu Marculescu. Prediction-based flow control for network-on-chip traffic. In *Design Automation Conference*, pages 839–844, 2006.
- [66] Miloš Panić, German Rodriguez, Eduardo Quiñones, Jaume Abella, and Francisco J Cazorla. On-chip ring network designs for hard-real time systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 23–32, 2013.
- [67] M. K. Papamichael and J. C. Hoe. CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, pages 37–46, New York, NY, USA, 2012. ACM.
- [68] D. Park, S. Eachempati, R. Das, A. K. Mishra, Y. Xie, N. Vijaykrishnan, and C. R. Das. Mira: A multi-layered on-chip interconnect router architecture. In *2008 International Symposium on Computer Architecture*, pages 251–261, 2008.
- [69] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 120–129, Aug 2008.

- [70] Christian Paukovits and Hermann Kopetz. Concepts of switching in the time-triggered network-on-chip. In *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 120–129, 2008.
- [71] M Paulitsch, E Schmidt, C Scherrer, and H Kantz. Industrial applications. In *Time-Triggered Communication*, pages 331–388. CRC Press, 2018.
- [72] Zhiliang Qian, Da-Cheng Juan, Paul Bogdan, Chi-Ying Tsui, Diana Marculescu, and Radu Marculescu. A comprehensive and accurate latency model for network-on-chip performance analysis. In *2014 19th Asia and South Pacific Design Automation Conference*, pages 323–328. IEEE, 2014.
- [73] Dara Rahmati, Srinivasan Murali, Luca Benini, Federico Angiolini, Giovanni De Micheli, and Hamid Sarbazi-Azad. A method for calculating hard QoS guarantees for networks-on-chip. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 579–586, 2009.
- [74] Eberle A Rambo and Rolf Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *Design, Automation & Test in Europe Conference & Exhibition*, 2015.
- [75] V. Rantala, T. Lehtonen, and J. Plosila. Network on chip routing algorithms. 12 2008.
- [76] Aleksandr Yu Romanov. Development of routing algorithms in networks-on-chip based on ring circulant topologies. *Heliyon*, 5(4):e01516, 2019.
- [77] K. Sai Reddy and K. Vipin. OpenNoC: An open-source NoC infrastructure for FPGA-based hardware acceleration. *IEEE Embedded Systems Letters*, 11(4):123–126, Dec 2019.
- [78] M. Shaheen, H. Fahmy, and H. Mostafa. Modified CONNECT: New bufferless router for NoC-based FPGAs. In *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 424–427, Aug 2018.
- [79] Z. Shi, A. Burns, and L. Indrusiak. Schedulability analysis for real time on-chip communication with wormhole switching. *International Journal of Embedded and Real-Time Communication Systems*, 2010.
- [80] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip*, 2008.
- [81] Zheng Shi and Alan Burns. Improvement of schedulability analysis with a priority share policy in on-chip networks. 2009.
- [82] Zheng Shi and Alan Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *21st Euromicro Conference on Real-Time Systems*, pages 3–12. IEEE, 2009.
- [83] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.
- [84] R. A. Stefan, A. Molnos, and K. Goossens. dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up. *IEEE Transactions on Computers*, 63(3):583–594, March 2014.
- [85] Domițian Tămaș-Selicean, Paul Pop, and Wilfried Steiner. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems*, 51(1):1–35, 2015.

- [86] Domitian TamasSelicean, Paul Pop, and Wilfried Steiner. Timing analysis of rate constrained traffic for the ttethernet communication protocol. In *IEEE 18th International Symposium on Real-Time Distributed Computing*, pages 119–126. IEEE, 2015.
- [87] Tiler. Tile64tm processor. www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf, 2012.
- [88] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer. IDAMC: A NoC for mixed criticality systems. In *IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013.
- [89] Ling Wang, Xiaohang Wang, and Yadong Wang. ABDTR: Approximation-based dynamic traffic regulation for networks-on-chip systems. In *International Conference on Computer Design*, pages 153–160. IEEE, 2017.
- [90] S. Wasly, R. Pellizzoni, and N. Kapre. HopliteRT: An efficient FPGA NoC for real-time applications. In *International Conference on Field Programmable Technology*, pages 64–71, Dec 2017.
- [91] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. Worst case latency analysis for Hoplite FPGA-based NoC. Technical report, 2017.
- [92] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE*, 2007.
- [93] Q. Xiong, F. Wu, Z. Lu, and C. Xie. Extending real-time analysis for wormhole NoCs. *IEEE Transactions on Computers*, 66, Sep. 2017.
- [94] Qin Xiong, Zhonghai Lu, Fei Wu, and Changsheng Xie. Real-time analysis for wormhole NoC: Revisited and revised. In *International Great Lakes Symposium on VLSI*, pages 75–80. IEEE, 2016.
- [95] Wang Zhang, Ligang Hou, Jinhui Wang, Shuqin Geng, and Wuchen Wu. Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip. In *2009 WRI Global Congress on Intelligent Systems*, volume 3, pages 329–333. IEEE, 2009.
- [96] Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. Timing analysis of rate-constrained traffic in TTEthernet using network calculus. *Real-Time Systems*, 53(2):254–287, 2017.