UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Politècnica de València

Departamento de Informática
de Sistemas y Computadores

Head-of-Line Blocking Reduction in Power-Efficient Networks-on-Chip

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
(Computer Engineering)

Author

José Vicente Escamilla López

Advisor

Prof. José Flich Cardo

September 2017

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

ABP	**A**ctivate **B**uffers **P**ath
AM	**A**llocation **M**essage
AN	**A**ctivation **N**etwork
ANN	**A**rtificial **N**eural **N**etwork-based
AsAP	**As**ynchronous array of **s**imple **P**rocessors
AVADA	**A**djustable **V**C **A**ssignment with **D**ynamic VC **A**llocation
BAHIA	**B**urst-**A**ware **H**ead-of-Line blocking **I**njection **A**voidance
BET	**B**reak **E**ven **T**ime
BNN	**B**urst **N**otification **N**etwork
CaL	**C**ongestion **a**nd **L**atencies
CMP	**C**hip **M**ulti **P**rocessor
CNN	**C**ongestion **N**otification **N**etwork
CP	**C**ongested **P**oint
DBP	**D**eactivate **B**uffers **P**ath
DC-DC	**D**irect **C**urrent-**D**irect **C**urrent
DVFS	**D**ynamic **V**oltage **F**requency **S**caling
DM	**D**eallocation **M**essage
DMSD	**D**elay-based **M**ax **S**low **D**own
FVADA	**F**ixed **V**C **A**ssignment with **D**ynamic VC **A**llocation
GHz	**G**iga **H**ert**z**
HAT	**H**eterogeneous **A**daptive **T**hrottling
HoL	**H**ead-**o**f-**L**ine
HPC	**H**igh **P**erformance **C**omputing
HPRA	**H**otspot-**P**reventive **R**outing **A**lgorithm
HSD	**H**ot**s**pot **D**estined
HT	**H**igh **T**hreshold
ICARO	**I**nternal-**C**ongestion-**A**ware HoL-blocking **R**em**O**val
MPSoC	**M**ulti **P**rocessor **S**ystem **o**n **C**hip
LT	**L**ow **T**hreshold
MC	**M**emory **C**controller
MHz	**M**ega **H**ert**z**

ND	**N**otification **D**elay
NI	**N**etwork **I**nterface
NoC	**N**etwork-**o**n-**C**hip
NonHSD	**Non**Hot**S**pot **D**estined
NoRD	**No**de **R**outer **D**ecoupling
PAPM	**P**ath **A**ware **P**ower **M**anagement
PARS	**P**ath **A**ware **R**outing **S**cheme
PGC	**P**ower **G**ating **C**ontroller
PI	**P**olling **I**nterval
PI	**P**roportional **I**nteger
PLL	**P**hase **L**ocked **L**oop
PWM	**P**ulse **W**idth **M**odulation
RCA	**R**egional **C**ongestion **A**wareness
RECN	**R**egnional **E**xplicit **C**ongestion **N**otification
RP	**R**outer **P**arking
RP-A	**R**outer **P**arking-**A**ggresive
RP-Adp	**R**outer **P**arking-**Ad**a**p**tive
RP-C	**R**outer **P**arking-**C**onservative
RSO	**R**equest **S**witch **O**ff
SAT_THR	**SAT**uration **THR**eshold
SoC	**S**ystem-**o**n-**C**hip
TBG	**T**ime **B**etween **G**enerations
TooT	**T**urn-**o**n **o**n **T**urn
UNSAT_THR	**UNSAT**uration **THR**eshold
V&F	**V**oltage & **F**requency
VFI	**V**oltage **F**requency **I**sland
VC	**V**irtual **C**hannel
VCO	**V**oltage **C**ontrolled **O**oscillator
VN	**V**irtual **N**etwork
VR	**V**oltage **R**egulator

Abstract

Nowadays, thanks to the continuous improvements in the integration scale, more and more cores are added on the same chip, leading to higher system performance. In order to interconnect all nodes, a network-on-chip (NoC) is used, which is in charge of delivering data between cores. However, increasing the number of cores leads to a significant power consumption increase, leading the NoC to be one of the most expensive components in terms of power. Because of this, during the last years, several mechanisms have been proposed to address the NoC power consumption by means of DVFS (Dynamic Voltage and Frequency Scaling) and power-gating strategies. Nevertheless, improvements achieved by these mechanisms are achieved, to a greater or lesser extent, at the cost of system performance, potentially increasing the risk of saturating the network by forming congested points which, in turn, compromise the rest of the system functionality. One side effect is the creation of the "Head-of-Line blocking" effect where congested packets at the head of queues prevent other non-blocked packets from advancing. To address this issue, in this thesis, on one hand, we propose novel congestion control techniques in order to improve system performance by removing the "Head-of-Line" blocking effect. On the other hand, we propose combined solutions adapted to DVFS in order to achieve improvements in terms of performance and power. In addition to this, we propose a path-aware power-gating-based mechanism, which is capable of detecting the flows sharing buffer resources along data paths and perform to switch them off when not needed. With all these combined solutions we can significantly reduce the power consumption of the NoC when compared with state-of-the-art proposals.

Resumen

Hoy en día, gracias a las mejoras en la escala de integración cada vez se integran más y más núcleos en un mismo chip, mejorando así sus prestaciones. Para interconectar todos los nodos dentro del chip se emplea una red en chip (NoC, Network-on-Chip), la cual es la encargada de intercambiar información entre núcleos. No obstante, aumentar el número de núcleos en el chip también conlleva a su vez un importante incremento en el consumo de la NoC, haciendo que ésta se convierta en una de las partes más caras del chip en términos de consumo. Por ello, en los últimos años se han propuesto diversas técnicas de ahorro de energía orientadas a reducir el consumo de la NoC mediante el uso de DVFS (Dynamic Voltage and Frequency Scaling) o estrategias basadas en "power-gating". Sin embargo, éstas mejoras de consumo normalmente se obtienen a costa de sacrificar, en mayor o menor medida, las prestaciones del sistema, aumentado potencialmente así el riesgo de saturar la red, generando puntos de congestión que, a su vez, comprometen el rendimiento del resto del sistema. Un efecto colateral es el "Head-of-Line blocking", mediante el que paquetes congestionados en la cabeza de la cola impiden que otros paquetes no congestionados avancen. Con el fin de solucionar este problema, en ésta tesis, en primer lugar, proponemos técnicas novedosas de control de congestión para incrementar el rendimiento del sistema mediante la eliminación del "Head-of-Line blocking", mientras que, por otra parte, proponemos soluciones combinadas adaptadas a DVFS con el fin de conseguir mejoras en términos de rendimiento y energía. Además, proponemos una técnica de "power-gating" orientada a rutas de datos, la cual es capaz de detectar flujos de datos compartiendo recursos a lo largo de rutas y apagar dichos recursos de forma dinámica cuando no son necesarios. Con todas éstas soluciones combinadas podemos reducir el consumo de energía de la NoC en comparación con otras técnicas presentes en el estado del arte.

Resum

Hui en dia, gràcies a les millores en l'escala d'integració, cada vegada s'integren més i més nuclis en un mateix xip, la qual cosa millora les seues prestacions. Per tal d'interconectar tots els nodes dins el xip es fa ús d'una Xarxa en Xip (NoC; Network-on-Chip), la qual és l'encarregada d'intercanviar informació entre els nuclis. No obstant això, incrementar el nombre de nuclis en el xip també comporta un important augment en el consum de la NoC, la qual cosa fa que aquesta es convertisca en una de les parts més costoses del xip en termes de consum. Per això, en els últims anys s'han proposat diverses tècniques d'estalvi d'energia orientades a reduir el consum de la NoC mitjanant l'ús de DVFS (Dynamic Voltage and Frequency Scaling) o estratègies basades en "power-gating". Malgrat això, aquestes millores en les prestacions normalment s'obtenen a costa de sacrificar, en major o menor mesura, les prestacions del sistema i augmenta així el risc de saturar la xarxa al generar-se punts de congestió, que al mateix temps, comprometen el rendiment de la resta del sistema. Un efecte col-lateral és el "Head-of- Line blocking", mitjanant el qual, els paquets congestionats al cap de la cua, impedixen que altres paquets no congestionats avancen. A fi de solucionar eixe problema, en aquesta tesi, en primer lloc, proposem noves tècniques de control de congestió amb l'objectiu d'incrementar el rendiment del sistema per mitjà de leliminació del "Head-of- Line blocking", i d'altra banda, proposem solucions combinades adaptades a DVFS amb la finalitat daconseguir millores en termes de rendiment i energia. A més, proposem una tècnica de "power-gating" orientada a rutes de dades, la qual és capaç de detectar fluxos de dades al compartir recursos al llarg de les rutes i apagar eixos recursos de forma dinàmica quan no són necessaris. Amb totes aquestes solucions combinades podem reduir el consum d'energia de la NoC en comparació amb altres tècniques presents en l'estat de l'art.

Acknowledgements

I would like to thank my parents, my sister and Tatiana who gave me their support. To my colleages and friends in the GAP group for those great moments we had in the laboratory. To Pedro J. García who contributed in most of the papers of this thesis and to all professors of the DISCA department.

I also want to specially thank Mario R. Casu who was my advisor during my internship in Torino, a great professor and a great person (and with a lot of patience).

And, finally, I want to thank even more specially to José Flich, my advisor, who put up with me for several years. Thanks for his invaluable support and knowledge. The best advisor I could had.

Chapter 1

Introduction

Nowadays, our society relies and depends on the never ending need of higher computing power and capacity. Computing power demand increases dramatically every year. New, never thought, emerging multimedia applications in the personal computer and embedded systems landscape and the need of High Performance Computing (HPC) for solving challenging and complex problems pose more and more computational demands while requiring to keep costs low in terms of power.

In order to meet these tight and conflicting requirements, in the last years, computing architecture realm has suffered a radical change in its paradigm. Traditionally, performance improvements in microprocessors have been achieved by improvements over the architecture (multitasking, cache memories, etc...), but, also by taking benefit of system's clock increase, which causes a high direct impact in system performance. For instance, firsts 80286-based[2] processors accounted with system's clock frequency in the order of few MHz's. In contrast, in 15 years processors raised the frequency to the order of GHz's. Increasing the frequency is the straight way to increase processors throughput. However, in the last years, clock frequency has reached its feasible limits. As seen in Equation 1.1, power depends roughly quadratically with the frequency, therefore, it is evident that to exceed the frequency above a given value will make the dissipated power to reach unfeasible values. Several solutions have been used in order to increase the clock frequency as much as possible. As an example, processor cooling systems have evolved from simple aluminum heat sinks with no fan to modern liquid cooling systems. However, advanced cooling systems are expensive and consume huge amounts of power. Therefore, associated costs become unaffordable in production platforms such as HPC systems.

$$P = CV^2 f \tag{1.1}$$

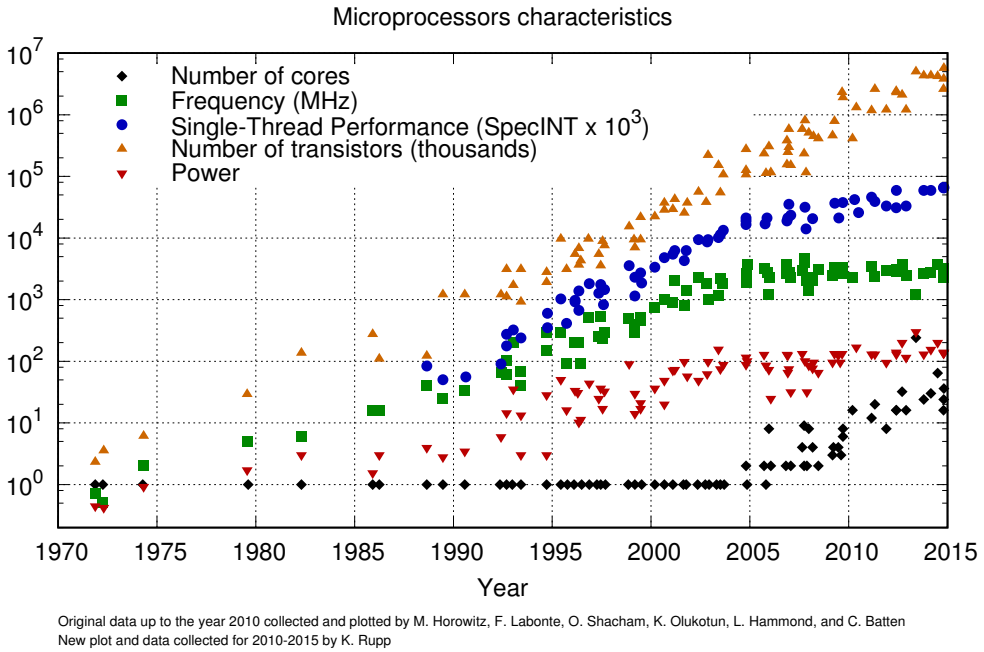


FIGURE 1.1: Microprocessors specifications timeline.

Because of the unfeasibility of speeding processors up by increasing the frequency, recently, chip manufacturers set the frequency increase strategy aside and opted to increase performance by means of implementing more cores in the same chip, which are termed chip multi-processors (CMPs). The idea behind this new strategy consists in, instead of relying on a big and complex monolithic processor running at high frequencies, to design simpler processors and physically replicate them several times while running at lower frequency values. In this way, applications can be mapped into different cores, hence, allowing them to run completely in parallel. This clearly implies a significant improvement in performance and power efficiency due to the benefits of parallelism. Due to the use of more power-efficient cores, a set of those cores can lead to same performance levels (or higher) for the same power budget in monolithic processors.

In Figure 1.1 we can see the evolution of key microprocessor parameters since 1970. As seen, until 2005, manufacturers kept increasing clock frequency for single-core processors. In 2005, the clock frequency reached the top value, after which, the clock frequency has been kept roughly constant while the increased factor has been the number of cores, causing only a slightly power consumption increment.

Currently, multi-core architectures are being used extensively with designs implementing typically from 2 to 24 cores. However, in order to increase parallelism and take even more advantage of the multi-processor paradigm, the trend is to move forward in this approach by adding tens, hundreds or even thousands of cores. These processors are called many-core processors.

Despite these efforts to control power consumption by means of the multi-core/many-core paradigm, power consumption still represents a key and difficult challenge, due to the current trend of implementing more and more cores in the same chip in order to satisfy both higher performance in HPC and to optimize the limited batteries lifetime in the emerging market of portable devices such as smart phones. However, one side benefit when using the multi- and many-core approach is that each core can be optimized and tailored with specific technologies to save power and, thus, to become more energy efficient. An example is the use of core level power-saving mechanisms such as DVFS[3] or power-gating[4], which are currently commonly used to achieve better power-efficiency results.

The multi-core and many-core paradigm is not only restricted to CMPs. CMPs are typically composed of several general-purpose microprocessors interconnected in order to run any application in any of their nodes. This could be seen as the evolution of the general-purpose microprocessor. Similarly, Systems-on-Chip (SoC) consist in a complete and specific system integrating on the same chip most of the components required (cores, encoders, specific function modules, memories, ...). As integration scale continued its evolution, SoCs, similarly to CMPs, evolved to MPSoCs (Multi-Processor System-on-Chip), in which, regular SoCs implement several processors to take advantage of the multi-core approach.

Both CMPs and MPSoCs need an interconnect fabric in order to work. This on-chip network[5][6], termed network-on-chip or NoC, is necessary in order to support the internal traffic between components in the same chip. In CMPs, caches interchange data blocks and commands (coherence traffic) and also access to external memory modules. Therefore, due to the type of traffic it transports, the NoC must be extremely fast and capable of serving data at very low latencies, otherwise the overall chip performance will be negatively affected. However, due to the intrinsic design restrictions inside the chip, the NoC must be carefully designed according to very tight constraints in terms of area and power. In current multi-core chip designs, since only a few nodes are typically implemented, such processors usually rely on simple buses or rings, as in the case of the IBM Power8, shown in Figure 1.2. Bus and ring topologies are simple and relatively inexpensive. However, since all nodes connected to the network share the same physical media, they do not scale in performance with the number of interconnected nodes, since the media can be used by only one node at a time. Therefore, to make many-core processors feasible, other network designs must be used in order to allow concurrent communication between all nodes. In this sense, currently, point-to-point mesh network topologies are emerging as the most popular interconnect strategy in many-core systems. Specifically, as microprocessors are manufactured over a 2D silicon substrate, 2D meshes fit naturally well in the floorplan. An example of this network topology is the Tilera TILE-Gx72 platform[7] with 72 cores shown in Figure 1.3 which is provisioned with 5 completely independent 2D mesh networks each one intended to a specific type of traffic.

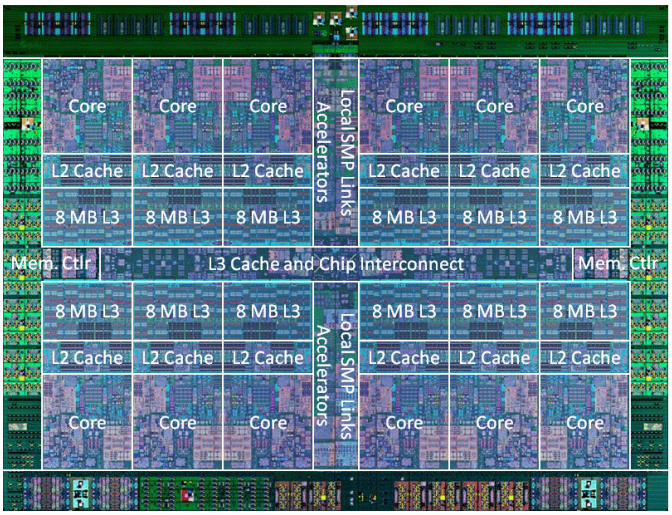


FIGURE 1.2: IBM Power 8 CMP chip.

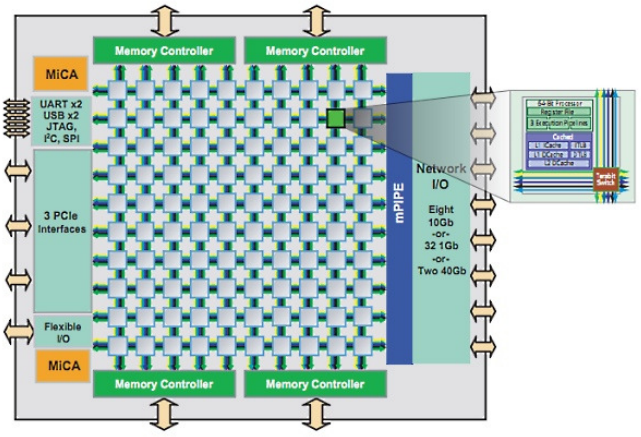


FIGURE 1.3: TILE-Gx72 platform provided with 72 tiles.

NoCs are being deeply researched for the last 15 years. These NoCs adopted many design styles and methods from networks designed for HPC systems. Indeed, these NoCs are not exempt from well known problems such as network routing deadlock, fault-tolerance designs, network contention and network congestion, to name a few. Also, the increase in number of computational units in CMPs and MPSoCs, makes the network utilization to increase, therefore, its design increases in complexity to increase its performance, leading to an increase in the power consumed by the NoC infrastructure as well. At the beginning of the concept, NoCs were used to provide connectivity between only a few nodes (from 2 to 8). For this number of nodes, the impact of the NoC power consumption is moderated compared with the overall chip power consumption. However, as the number of nodes in the chip increases, the NoC power consumption increases significantly as well, representing a higher portion of the overall chip power consumption. In fact, some authors have demonstrated that the NoC power consumption reaches up to 28% [8] of the overall 80-cores chip power consumption, representing one

of the most power-hungry parts of the chip. Because of this, several works focused on the NoC power consumption by extending the use of power control techniques -typically focused to control the power consumption of cores- to the NoC infrastructure.

Among all power savings techniques, currently, the most extended strategies come in two flavors: DVFS (Dynamic Voltage Frequency Scaling) and power-gating. Both mechanism aim to save power but through different approaches.

DVFS works by adjusting the voltage and the clock frequency in order to increase or decrease the system performance. The idea behind DVFS is to adapt the system performance to meet the current application requirements so that the more the frequency is decreased, the more power-saving is achieved according to Equation 1.1.

DVFS techniques, when applied to cores, rely on different metrics representing the core utilization, so that the V&F (Voltage and Frequency) controller is aware of it in order to decide whether to increase/decrease the core frequency. In order to drive the NoC V&F, these metrics become useless since the network utilization may not be directly related with cores activity. Therefore, in order to drive NoC V&F, new metrics and algorithms were conceived. Typically, these metrics are related to the buffers utilization, data-rates or message latencies. However, to choose a metric or a given set of metrics is not trivial. NoCs activity can be complex due to the intrinsic unpredictability nature of the traffic flow and its interactions. Indeed, an efficient DVFS approach applied to NoCs is still a challenge.

Typical DVFS techniques works by decreasing the voltage and frequency in case of resources to be underutilized, thereby achieving power-savings. Similarly, frequency and voltage are increased when detecting that some resources are fully utilized at their current frequency, leading to a higher performance but at the cost of power consumption. The key idea is to tune the frequency and voltage to achieve a balance in which the performance loss is acceptable while achieving a significant power-saving. Achieving this balance point is not easy, specially when applied to NoCs. DVFS always implies potential performance loss since, to decrease the clock frequency, even assuming a single flow crossing the network, always implies a linear increment to the end-to-end latency.

Furthermore, if the DVFS policy is not well designed or running under specific circumstances (for instance, under bursty traffic or unbalanced traffic patterns), DVFS may wrongly decide to decrease frequency, saturating the network and, thus, favoring the creation of congestion spots. This may revert into a significant overall system performance degradation for relatively small systems, but in large NoCs, which are more sensitive to saturation, the system will be more affected by this side effect. Thus, voltage and frequency control becomes more critical as network increases in size.

One of the main issues of DVFS is related to its granularity. DVFS was initially designed to support a single V&F domain. This means that V&F is set for all nodes in the system,

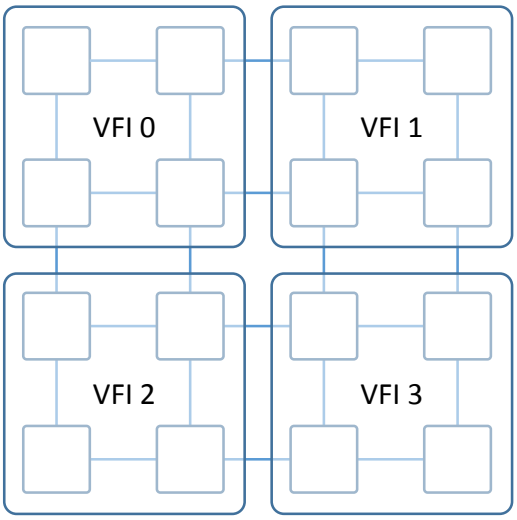


FIGURE 1.4: DVFS voltage and frequency islands in a 16 Mesh system.

making all nodes to work at the same voltage and frequency. This could lead in V&F suboptimal adjustment so that while a given set of nodes in the network would require to rise the frequency to meet the application traffic requirements, other nodes could be completely underutilized, causing a waste of power in case of raising the frequency. To face this issue, V&F islands (VFIs) were proposed. In this way each VFI is driven by an independent V&F controller so that each VFI is responsible of monitoring the activity of the nodes belonging to each VFI and setting its voltage and frequency accordingly and independently of other VFIs. In Figure 1.4 we can see an example of a 4x4 2D mesh network with 4 VFIs. This approach effectively increases its granularity, meeting more accurately the needs of the system. However, each VFI requires its own V&F regulator, which could be very expensive and completely unfeasible beyond a given granularity level, as stated in [9].

As stated previously, an alternative approach for saving power is *power-gating*. This mechanism essentially consists in powering off unnecessary devices or parts of them, depending on the mechanism granularity. Concerning NoCs, power-gating works by monitoring the network in order to switch off unused routers or links. However, this mechanism poses drawbacks. For instance, switching off a complete router may affect network connectivity leading to parts of the system being disconnected. To solve this, specific parts of routers can be switched off independently, guaranteeing connectivity, since the rest of the router parts are kept in service.

Power-gating mechanisms usually require a centralized controller in order to collect metrics from the system. Once these metrics are collected, the controller decides the routers (or parts of them) to be switched off/on and sends the corresponding signals to switch them on/off. This means that, the more the granularity, the more complex becomes the control over the parts to be switched off.

Other aspects of power-gating mechanisms are related to the delays and overheads. Switching off/on a device induces a penalty in terms of delay and power mainly due to electronic limitations when rising the voltage. This means that the power-gating control must be carefully designed in order to switching on/off a given device properly. Otherwise, it could even cause an increase power consumption rather than power savings. In terms of latency, additionally to the inherent delay caused by scaling up the voltage, switching resources off also may lead to large transmission delays as some components required by incoming traffic could not be switched on in time for serving this traffic. This also leads to potential congestion effects that can spread over the network, similar to the ones potentially originated by DVFS techniques, leading to a serious decrement of system performance.

As previously stated, both DVFS-based techniques and power-gating techniques may cause performance degradation at NoC level, making the network potentially weak and prone to congested situations. Indeed, traffic patterns in NoCs are typically characterized by their irregularity[10] and burstiness[11]. Irregular traffic is even more apparent in new heterogeneous systems[12][13]. In this scenario, we can classify congested traffic and non-congested traffic both coexisting on the same NoC. This mixture is the perfect recipe for generating the Head-of-Line blocking(HoL)[14] effect. HoL-blocking emerges when congested data flows share the same buffers than non-congested ones. As seen in Figure 1.5, if a congested flow reaches the head of a queue and, due to output resources congestion, is not able to be forwarded to the next router, it will get blocked (*red* flow in the cited figure). In the same figure, the following message in the FIFO queue corresponds to a non-congested flow which requests a non-congested output port, which has all resources available to forward the non-congested flow. However, since the congested message keeps blocked waiting for its resources, the non-congested flow gets blocked *unnecessarily*. This effect of blocking messages even having requested resources available is known as HoL-blocking.

Several proposals in the literature deal with this harmful effect, being the most significant one the RECN mechanism[15]. RECN is innovative in the sense that it is the first work that considers congestion not to be a real problem by itself. Indeed, congestion leads to severe HoL blocking effects between congested and non-congested flows, and this is what makes congestion to affect performance negatively. If the HoL blocking effect caused by congestion is totally removed then performance is not affected at all and congestion becomes harmless. To solve the HoL blocking effect, RECN uses a sophisticated and dynamic mechanism implemented in routers. Unfortunately, the complete implementation of RECN on NoCs is unfeasible given the current constraints of area and power imposed in NoCs. This means the RECN mechanism can not be applied directly on NoCs. However, its philosophy (separating non-congested flows from congested ones) can be considered for achieving effective performance in NoCs. This sort of congestion management becomes even more important in environments provided with DVFS and

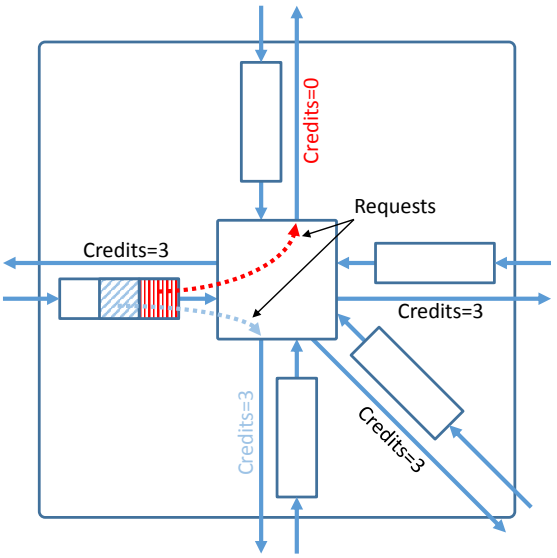


FIGURE 1.5: Head-of-Line blocking.

power-gating techniques due to the performance degradation caused by them, which could generate congestion. This thesis addresses this fact and focuses on an integral approach to deal with the design of NoCs with power efficient techniques while smartly addressing congestion situations.

In this thesis we provide efficient and effective methods that address power saving while improving performance levels of the NoC and applications running on the system. In particular, as a novelty of this thesis, we address the two problems of congestion management and power-saving providing separated solutions but also combined ones. Indeed, one central contribution of the thesis is the achievement of a mechanism able to deal with congestion while reducing power consumption by using power minimization strategies (namely DVFS and power gating). As stated previously, those techniques are prone to create congestion spots as they affect the operational frequency of the system and its resources availability. Therefore, combining them and taking into account both congestion management and power minimization strategies is worth to be analyzed and may lead to more effective solutions.

More specifically, Figure 1.6 shows the different contributions of the thesis. As a first step, we propose two different (but related) congestion control mechanisms: BAHIA and ICARO. With these mechanisms congested traffic is logically and dynamically separated in different queues in the network, guaranteeing the side effects of congested traffic over non-congested one to be avoided. This is mainly the HoL-blocking effect where a congested message prevents non-congested ones from advancing. Although this way of actuating does not eliminate congestion (but eliminates the side effects) is more effective as it has been proven in the past in [15]. Indeed, this novel approach (removing side effects of congestion rather than eliminating congestion) has never been directly applied to NoCs. The difference between the two methods relies on the location where congestion

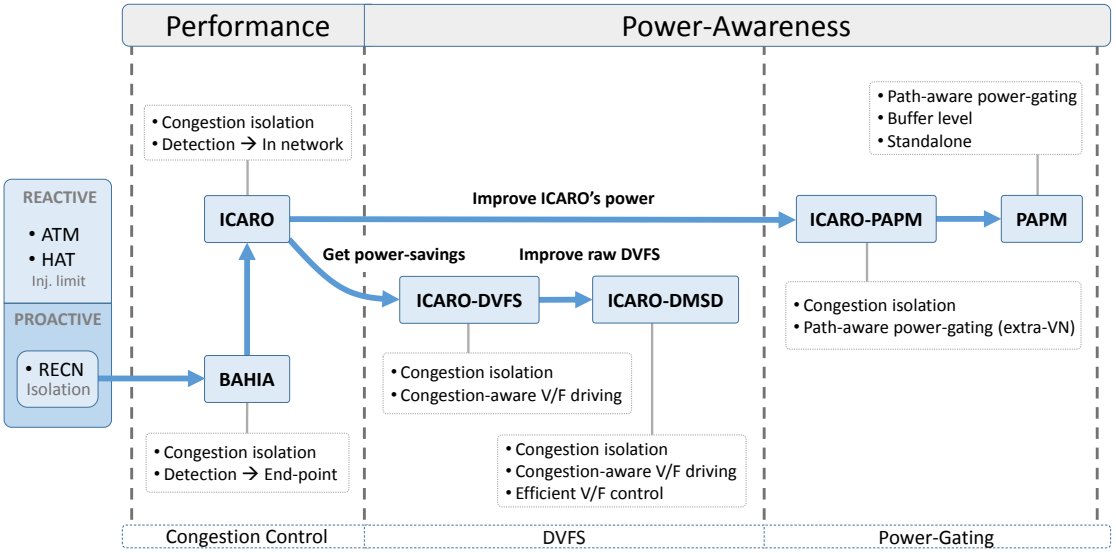


FIGURE 1.6: Thesis outline.

is detected, which is at the end-nodes for BAHIA and at each router in ICARO. In this thesis, we propose an effective and efficient implementation of a congestion-management strategy focusing on HoL-blocking removal.

Then, following the most evolved congestion control mechanism (ICARO), we adapt it to a system design where multiple V&F domains exist and where DVFS techniques are applied to each domain. Specifically, we deploy the ICARO-DVFS strategy. In this strategy, ICARO is used to assist DVFS by allowing DVFS performance metrics to be collected following three different approaches allowed by the integration of ICARO with DVFS. According to these three different collection strategies, we present three different approaches to integrate DVFS with ICARO, each focusing on an optimization parameter (power consumption, message latency, or a combination).

On a similar trend, we adapt ICARO to the DMSD method. DMSD stands for Delay-based Max Slow Down (DMSD) which is a DVFS policy that actuates based on the delay of the message flows from every end-node. Basically, DMSD guarantees a latency target for each communicating flow and actuates on the DFVS controller in order to minimize power consumption while guaranteeing such latency target. In the new proposal (ICARO-DMSD) we integrate the ICARO mechanism to guarantee the isolation of congested background traffic and letting the DMSD strategy to operate only on sensitive traffic. In other words, we isolate congestion to let DMSD be still effective under conflictive traffic patterns. These two methods (ICARO-DFVS and ICARO-DMSD) provide an overview how congestion management strategies can be applied to DVFS related techniques.

Finally, and to embrace a larger scope in power minimization strategies, we address power-gating strategies. Thus, we propose the ICARO-PAPM mechanism where we

combine ICARO with a novel path-aware power-gating mechanism. Basically, PAPM (proposed as a contribution in this thesis as well) selectively powers on and off paths based on the traffic activity related to that path. If a path is not used for a period of time then the path is powered off and all the associated buffers along the path are potentially powered down. However, as router buffers are shared we need to have a strategy to avoid powered down buffers on active paths. In PAPM we address this issue. The ICARO-PAPM mechanism let's an effective use of buffers in terms of power. Indeed, as ICARO avoids congestion by separating the congested traffic from the non-congested one, as long as contention is not present, these inactive buffers can be powered off. ICARO-PAPM will manage those inactive buffers from a power consumption perspective. Notice that we propose both, the PAPM mechanism and the combined form that leads to ICARO-PAPM.

To summarize, our contributions in this thesis are:

- BAHIA: A congestion control mechanism based on congested traffic detection **at the end-nodes** and its isolation to avoid the HoL-blocking
- ICARO: A congestion control mechanism that detects congestion **at routers** (more accurate) and isolates it.
- ICARO-DVFS: Combination of DVFS with ICARO to improve the efficiency of DVFS provided systems. Three solutions are proposed aiming to improve different parameters: Power saving, performance, or a balanced combination.
- ICARO-DMSD: Combines ICARO with DMSD (latency-driven DVFS) to improve DVFS-based scenarios by preventing from overdriving the frequency under hotspot and/or bursty traffic patterns while keeping the latency bounded.
- ICARO-PAPM: Combines PAPM with ICARO in order to improve ICARO power savings by switching off the *extra virtual network* used by ICARO when not needed while improving system performance.
- PAPM: A path-aware power management mechanism consisting in power-gating router buffers based of flows paths.

1.1 Thesis Outline

Following the rules of *Universitat Politècnica de València*, this thesis has been written as a compendium of articles and is structured as follows:

- In Chapter 2 the background of this thesis is described as well as related work. Although subsequent chapters will include related work sections, we provide in

this chapter a complete and integrated background and related work description in order to provide a unified view to the reader.

- In Chapter 3 we put together all the descriptions of this thesis proposals, together with their evaluations and an assessment of their similarities, differences and complementarities. The goal of this chapter is to ease the understanding of the proposals and to let the reader be focused on the proposals.
- From Chapter 4 to Chapter 9 the compendium of papers are arranged as follows:
  - In Chapters 4 and 5 we describe the basic congestion control mechanisms, namely BAHIA and ICARO, respectively.
  - In Chapters 6 and 7 we propose two approaches to combine ICARO with DVFS-based mechanisms to efficiently manage power consumption by isolating congested traffic.
  - In Chapters 8 and 9 a path-aware power-gating mechanism is proposed to improve ICARO power consumption and adapted to general NoCs respectively.
- Finally, in Chapter 10 we expose the conclusions, discuss future work derived from this thesis and enumerate all conferences in which the articles of this thesis have been published in.

Chapter 2

Background and Related Work

In this chapter, we collect basic concepts required to fully understand all proposals described in this thesis. The content of this chapter provides the background related to networks-on-chip, power-saving mechanisms and congestion management. Along the description, we also introduce some of the existing work in order to provide an up-to-date view of the state-of-the-art related to this thesis.

First, we center our description on congestion management and then on power-saving techniques.

2.1 Congestion Management

NoCs must be capable of delivering traffic extremely fast but, at the same time, very efficiently in terms of power. This, by itself, represents a challenge. However, on-chip designs also pose very tight area constraints as on-chip networks must be carefully designed in order to minimize physical resources required to perform successfully. These tight constraints may lead to insufficient provisioning of resources, causing network saturation, thereby generating congestion which degrades network performance and causes the system performance to degrade as well.

Congestion is defined as the effect of suffering contention along the time, therefore, to understand how congestion arises in the network, first contention background must be described to fully understand its origin and behavior.

Contention in a network arises when an output port is not capable of serving all data flows requesting the output port as the incoming flow bandwidth exceeds the output port bandwidth. In other words, contention occurs when the sum of all incoming rate flows requesting a given output port exceeds the output port maximum capacity, as formulated in Equation 2.1

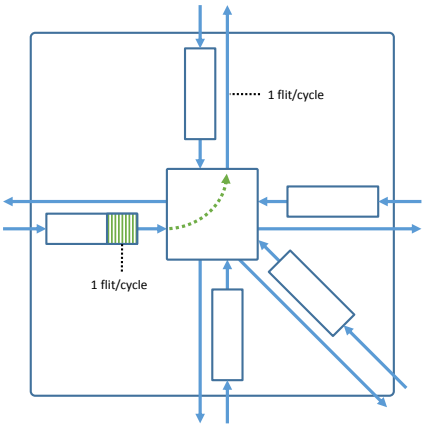


FIGURE 2.1: One flow forwarded at high data rate

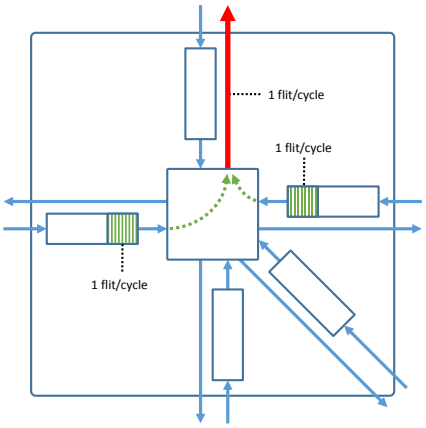


FIGURE 2.2: Two flows causing contention

$$\sum_{ip=0}^{r} \lambda_{ip} > \mu_{op}, \forall ip \neq op \tag{2.1}$$

where r is the number of router ports, λ is the arrival data rate for the input port denoted by ip and μ represents the maximum data rate the output port (op) is able to serve. The output port capacity, typically, is equal to the maximum arrival data rate at the input port. Because of this, a single flow will never exceed an output port capacity, thereby will never be able to cause contention as shown in Figure 2.1, thus will never generate congestion as well. Due to this, contention is only generated when two or more input ports request the same output port as shown in Figure 2.2, where the sum of both flows data rate exceeds the output port capacity, making both flows to be slowed down.

Contention can occur without degrading significantly system's performance. If contention occurs sporadically, router buffers along the flow path will absorb accumulated traffic in a lesser or greater extent, depending on the router buffer depth, avoiding excessive contention effects, thereby causing negligible network turbulence. However, in case of contention to last for moderate or large amounts of time, buffers will quickly filled, triggering the flow control to stop incoming flows from upstream routers, which may cause those router buffers to be filled as well, thereby propagating this effect to the rest of routers, spreading contention over the network starting from the first contended router and creating branches due to the interaction of other data flows with the congested flow as shown in Figure 2.3. When this happens, the network is *congested*.

Due to the negative impact of congestion, and also to the growing popularity of NoC-based systems, the number of proposals for congestion management in NoCs has quickly increased during the last years. Although some congestion-management mechanisms have been proposed for bufferless NoCs, such as the one presented in [16], we focus on

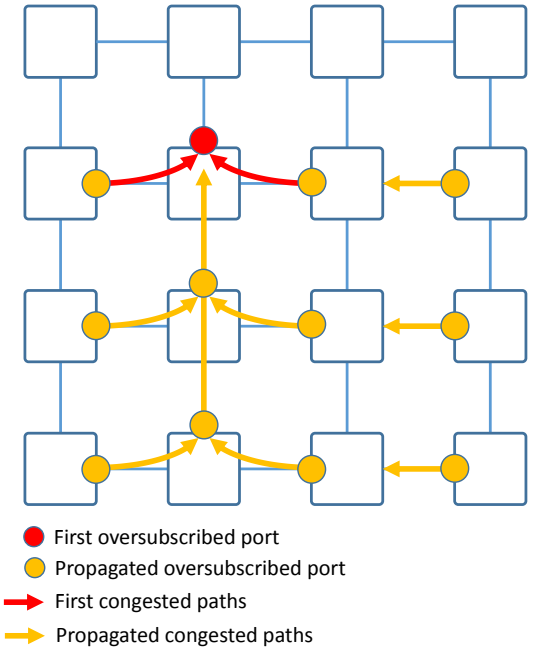


FIGURE 2.3: Congestion propagation.

solutions intended to buffered NoCs, as our proposals have been designed for this type of NoCs architectures.

Some works try to eliminate congestion by means of *injection throttling*, which corresponds to one of the "classical" approaches to congestion management. *Injection throttling* essentially consists in to decrease the injection rate at sources to reduce the network load, thereby alleviating or removing congestion. In [17], authors propose a self-tuning injection throttling-based mechanism which monitors the whole network buffers occupancy to compare them with a self-tuned threshold which is in charge of deciding whether to throttle traffic or not. Similarly, in [18] authors proposed HAT, a more sophisticated injection throttling mechanism. It works by classifying applications according to the intensity of its traffic generated at sources. Then uses this classification in order to decrease the injection rate only for the traffic generated by high network demanding applications. Other approaches, instead of evaluating directly metrics collected from the network to know the current network status, rely on more sophisticated mechanisms in order to compute predictions. For instance, in [19], authors propose an end-to-end flow control mechanism based on prediction-models to control the injection rate at the source node. Predictions are computed in every router using its state and its neighbors state. In order to exchange the necessary data for computing the prediction, routers implement additional wires interconnecting them. However, all injection throttling-based mechanisms suffer from the same issue. Like any control strategy based on closed-loop theory, may present performance oscillations and become inefficient if source nodes react too late.

NoCs, as well as any other network, need to implement routing algorithms to deliver traffic from a given source to any destination. NoCs in CMPs typically follow a 2D regular mesh topology and are subjected to the simplicity paradigm to save area and power. Thus, NoCs are typically provided with deterministic routing (Dimension Order Routing, DOR) due to its simplicity and effectiveness. However, in order to deal with congestion, some works propose to replace DOR by dynamic routing policies which collect metrics from the network to offer alternative paths to route around congested areas, thus increasing network performance. This *adaptive routing* approach is the basis of solutions like RCA[20], which uses a composition of multiple global metrics collected by means of piggybacking data into messages from the whole network to decide at each router output port which message is forwarded through, so hotspots are avoided. Similarly, in [21] authors propose to collect congestion information from the whole network and to take routing decisions based on network status. However, both proposals present similar potential drawbacks. In both mechanisms the congestion information is collected by piggybacking the links status into the packets header, which, in case of heavy-congestion situations, both mechanisms may collapse since the information used to avoid congested areas is aggregated in the same messages that are congested, which slows down the metrics delivery, causing the mechanisms to react too late. Additionally, adapting the routes to avoid hotspots may result in moving the location of such hotspots from one place to another, so the problem would remain unsolved. Moreover, avoiding hotspots may be impossible if all the congested flows have the same target (e.g. the memory controller).

Piggybacking metrics is an area-efficient strategy to deliver congestion-related data, but it may lead to wrong or too delayed corrective actions. To solve this, other proposals implement dedicated simple networks to avoid such delays. For instance, PARS, proposed in [22], uses a dedicated network for sending congestion metrics based on the buffer state at certain routers. Like RCA, PARS uses such metrics to select proper paths in order to avoid hotspots. Although in this case the information is sent through the dedicated network, the problems regarding unavoidable hotspots or "hotspot reallocation" may still appear. Similarly, in [23] authors propose a token-based flow-control mechanism which uses dedicated wires to send routers status information (token) which is used to take routing decisions and bypass router pipelines. However, this proposal is focused on reducing network latency by skipping routers pipelining, but not by facing congestion harmful effects.

Depending on the intensity and persistence of congested flows, congestion may be propagated very fast. Due to this, is key to detect it and react as fast as possible. However, due to its stochastic intrinsic characteristics, congestion detection becomes challenging. Most of the works rely on metrics related to the network or end-to-end latency. These metrics measure the time spent for traversing the network from the source to the destination and the time spent from the data allocation at NIs to its arrival to the destination.

Nevertheless, to keep track of these metrics typically requires these metrics to be delivered to the logic block in charge of reacting against congestion in case of being detected, which may take several cycles. Probably, when the congestion is intense enough to significantly affect the latency and, after the required time to deliver and evaluate those metrics, as stated in [17], it will be too late to react in time to avoid congestion from affecting the system performance.

Being aware of this, other proposals face the congestion caused mainly by hotspots by addressing it from a prediction-based approach. This is the case of [24], in which authors propose HPRA, a hotspot-formation prediction mechanism. HPRA uses an Artificial Neural Network-based (ANN) hardware that gathers buffer utilization data to predict the formation of hotspots. Then, HPRA classifies the traffic into two classes: hotspot-destined traffic (HSD) and non-hotspot-destined traffic (nonHSD). HSD traffic is throttled at source while the nonHSD traffic is routed avoiding paths containing hotspots routers. However, in the cases in which the ANN fails to predict hotspots, it may redirect traffic to an unpredicted hotspot, causing an even worse degradation of the system performance. Besides, HPRA suffers from the same metrics delivering issue of previously described RCA.

Most congestion control proposals are focused in dealing with congestion, either by acting over the injection rate or by dynamically routing traffic. However, regarding congestion control, in this thesis we focus our work from a different point of view. We claim that congestion is not a problem by itself but the real problem is the effects of congestion over non-congested traffic due to the HoL-blocking effect. Acting over congested traffic by reducing its rate is not an effective solution due to the oscillation effects described previously because the interaction between congested and non-congested flows is only alleviated, not solved, and to decrease the injection rate, even in a application-aware manner, deliberately decreases the application performance. Acting over congested traffic by detouring it (adaptive routing), can potentially reduce the interaction between congested flows and non-congested ones, but only because the availability of alternative paths. This strategy is not designed for this purpose, therefore, there is no guarantee of performing in that way.

Because of this, from our point of view, there is a need to change the paradigm to address congestion issues in NoCs. We show that an efficient HoL-blocking-avoidance mechanism must explicitly identify congested flows in order to isolate them completely and dynamically. A solution that follows a similar approach has been proposed in [25]. Actually, authors propose two policies to map traffic flows to VCs: FVADA and AVADA. Both proposals establish a correspondence between the output port requested on the router *x+1* and the output VC assigned in the router x (note this requires *lookahead routing*). The main difference between both policies is that FVADA establishes a direct and permanent correspondence between the requested output port and the assigned VC, while AVADA starts establishing a direct correspondence but later this

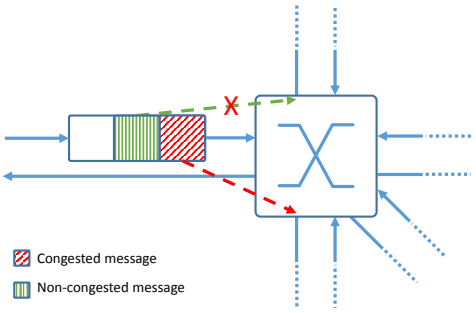


FIGURE 2.4: Congested message blocking non-congested message (HoL).

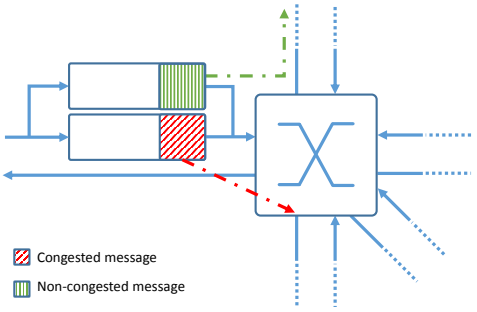


FIGURE 2.5: Congested message is isolated from the non-congested one.

correspondence can be dynamically adapted, based on the output port load, making use of a correspondence table (a CAM-based table). Note that, while FVADA is simpler to implement, it requires exactly as many VCs as the $router\_radix-1$ value, thus depending the number of required VCs on the router radix. Moreover, both policies require routers implementing lookahead routing and a credit-based flow-control in order to quantify the output port load and adapt their behavior when the load in a given VC is too high. Note that neither FVADA nor AVADA are actually aware of which traffic flows are contributing to a hotspot, as they only consider one hop (i.e. the next requested output port) in the path of the messages, while hotspots may be located further away. Thus, congested flows may still share queues with non-congested ones, thereby still causing HoL-blocking in some degree.

Following the HoL-blocking removing paradigm, a more convenient approach is proposed in [15], where authors propose RECN (Regional Explicit Congestion Management), a mechanism for isolating congested traffic for off-chip networks. Among the plethora of proposals for congestion management in off-chip networks, RECN can be considered as one of the most efficient as it completely prevents HoL-blocking while requiring a reduced set of queues. However, adapting the RECN basics to NoCs requires a very different way of implementing it, due to the tight limitations in area and power in this context. This is the starting point of the congestion side of this thesis. Under the paradigm proposed in RECN we propose a mechanism to deal with congestion by detecting congested flows and isolate them into special queues. In this way, interaction between congested flows and non-congested ones is avoided, hence removing the HoL blocking effect as following illustrated. In Figure 2.4 a router queue is depicted containing one congested message at the top of the queue blocking the non-congested one. Our proposal basically consists in adding a special queue intended to isolate congested traffic while the other queue stores non-congested one, as shown in Figure 2.5, thus allowing this traffic to be forwarded regardless the status of the congested traffic.

2.2 Power Saving

2.2.1 Dynamic Voltage and Frequency Scaling

To deal with the growing demand of performance, manufacturers traditionally opted by increasing the system clock frequency. This strategy has been a feasible solution to speed up processors. However, current silicon technology imposes restrictions to this approach since dissipated power scales exponentially with the frequency, making power consumption completely unfeasible at high frequencies. As seen in Equation 1.1, reducing the frequency and the voltage decreases the power consumption significantly. Because of this, one of the most extended strategies intended to decrease power consumption is DVFS[3] (Dynamic Voltage and Frequency Scaling). DVFS takes advantage of the resources underutilization to decrease the clock frequency opportunistically, which has a direct impact on the power consumption. However, reducing the clock frequency also decreases the system performance. Therefore, DVFS-based mechanisms analyze applications requirements and adapt the clock frequency in order to meet these requirements, thus saving power when requirements are low. Nevertheless, to decrease the frequency and voltage according to the application needs may generate additional issues. To determine when to increase or decrease the frequency is critical since wrong decisions adjusting the frequency may affect negatively to the performance and power consumption.

A typical implementation of DVFS is composed of a metric collector/evaluator and a voltage/frequency controller. The metric collector is the module in charge of collecting different metrics representing the system utilization. These metrics are evaluated to determine the relationship between applications requirements and system performance. The goal of DVFS is to adjust the V&F in order to meet as fast and as close as possible the system performance with the application requirements in order to achieve the maximum power savings without affecting significantly the application performance. Therefore, becomes key to determine accurately the application requirements through a proper metric (or metrics set). An example could be a mechanism that sends average end-to-end latencies from each node to the collector which, after receiving all latencies, evaluates whether the latencies are close to the network saturation point or not and triggers a frequency increase/decrease respectively.

The V&F controller corresponds to the physical driver which sets the network voltage and frequency. This controller is composed of the voltage regulator and the frequency driver. The voltage driver is typically implemented by means of a DC-DC buck converter which, at the same time is driven by a PWM/PFM controller as shown in Figure 2.6, setting the operating voltage according to the system needs. Regarding the frequency controller, early implementations assumed voltage-controlled oscillators (VCO) as the one implemented in the TI MSP430, which performs automatic and continuous frequency

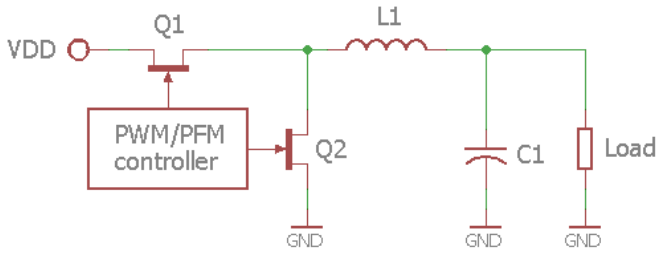


FIGURE 2.6: Voltage regulator.

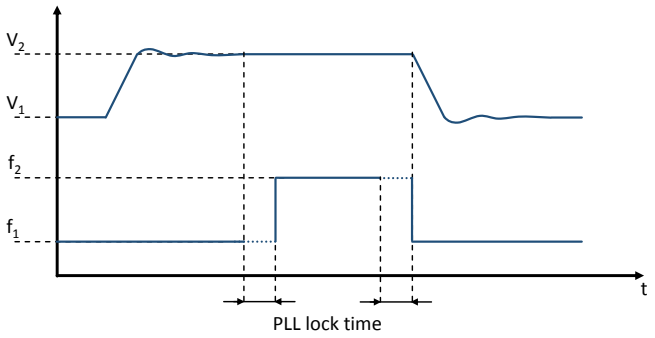


FIGURE 2.7: Voltage and frequency change process.

changes according to the operating voltage, allowing to keep the system running during all the voltage transition. However, this type of controllers are not commonly used in high-performance computing due to their frequency instability and inaccuracy at high frequencies. In turn, recent frequency implementations rely in PLLs due to their accuracy. However, in systems provided with PLLs to generate clock signals, first the voltage controller increases its value and waits until the voltage is stabilized to ensure the system stability and then the PLL changes its value to set the new frequency, which takes a lapse of time in the order of μs. To scale down the voltage and frequency, first the frequency is decreased and then the voltage is downscaled as described in Figure 2.7. PLLs, contrarily to the VCOs, must wait for the frequency to stabilize. This delay is called *PLL lock time*, and it is considered the main source of delay when raising the frequency since, during this time, devices powered by the controller are completely halted. These delays may affect negatively to the system performance, therefore, to abuse of V&F changes may affect to the system performance, increasing the risk of generating congestion in the NoC.

Voltage regulators can be implemented either on-chip or off-chip, depending on the requirements they must satisfy. As described in Table 2.1, on one hand, on-chip voltage regulators are characterized by their speed as they are able to switch in the order of tens of nanoseconds but they are very expensive in terms of area and its efficiency is lower than off-chip VRs[9]. On the other hand, off-chip regulators, are able to switch in the order of few milliseconds, which is two magnitude orders above on-chip VRs. However, off-chip VRs, as are implemented outside the chip, the area spent to implement them

Type	Speed	Area	Efficiency
On-Chip	Fast	High	∼80%
Off-Chip	Slow	None	∼90%

TABLE 2.1: Voltage regulators comparison.

represents not a real issue, hence they are able to deliver huge amounts of power and more efficiently to the chip.

As seen, current typical DVFS implementations incur in several μs of delay. Nevertheless, there are other implementations as the one performed in the AsAP processor[26], in which each processor can selectively switch from V_{ddHi} to V_{ddLo} and vice versa. As this is performed by switching the current power source from the V_{ddHi} power grid to the V_{ddLo}, V&F changes can be performed in the order of ns.

Regarding the metrics used in DVFS to evaluate the network performance, we dispose of several metrics as the buffers usage, received data-rate, end-to-end latency, etc. However, none of these metrics could be considered representative of the overall system state. Other factors like congestion, unbalanced traffic patterns could turn these metrics useless since may represent only a small portion of the overall system, potentially causing the frequency to increase or decrease due to a small part of the network traffic. To solve this, Voltage and Frequency Islands (VFIs) were proposed. VFIs create V&F (Voltage and Frequency) domains containing part of the network and controlled by a dedicated V&F controller. In this way, V&F of the nodes contained in each VFI are controlled independently of the rest of the system and network utilization metrics are collected only for these nodes. By doing this, V&F control granularity is increased so that the V&F can meet more precisely application requirements, thus improving power-savings and system performance. In this sense, most works drive the VFIs concept until its limits by increasing the granularity to the router and even to the link level[27][28][29][30][31][32]. These works, however, do not consider the overhead of having multiple voltage regulators and PLLs for the various NoC components, not to mention the latency penalty due to multiple clock-domain crossings. In fact, in [9], authors state that the area spent by VRs is proportional to the power to be delivered and it is estimated that it is necessary $2mm^2$ to deliver 1W of power. Therefore, to provide power enough to supply a core, it is necessary almost the same area that the area spent by the core itself, which is not practical[33].

Since on-chip and off-chip VRs, each one, exhibits advantages and drawbacks, a more feasible approach of increasing the VFIs granularity consists in to take advantage of both by designing a mixed approach as proposed in [9], in which both VR types are used.

However, we are closer to the view of other authors that consider more practical to have a single voltage and frequency domain for the whole NoC [34][35][36][37].

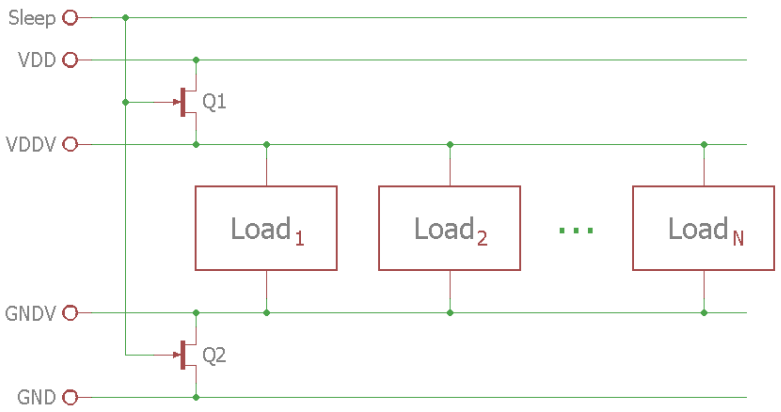


FIGURE 2.8: Power-gating implementation.

It is apparent that a fine-grain DVFS approach would lead to better power savings, but the implementation cost would be too high. For these reasons researchers explored a middle ground that we can classify as coarse-grain NoC DVFS, in which either multiple NoC planes (typically two planes) powered at different voltages and/or frequencies are used [38][39], or routers that can individually choose between only two voltages are employed [40].

2.2.2 Power-Gating

Advances in the integration scale not only come with the benefit of allowing to integrate more and more transistors in the same area but also with decreasing the overall power consumption due to the reduction of the capacitances of the circuitry. However, this helps to decrease the dynamic component of power consumption caused by signal toggles but it has negligible effects over the leakage power. As the technology goes further, the importance of the leakage power grows similarly, leading to more works addressing this challenge. The most extended mechanism to deal with the leakage power is power-gating, which essentially consists in powering off unused devices or components, thereby avoiding them to consume leakage power when they are not needed. A traditional power-gating implementation is shown in Figure 2.8. *Q1* and *Q2* are MOSFET transistors which are driven by the sleep signal coming from the power-gating controller. These transistors provide power from *VDD* and GND to the virtual power rails *VDDV* and *GNDV* respectively, which, in turn, supply power to the device or logic to be controlled.

To switch a device off and on is not for free. Due to the power-gating circuitry decoupling capacitances and other factors, to switch on/off a device incurs in power overheads that, in absence of power-gating mechanisms, would not exist. This power overhead depends on the circuitry capacitances but also exhibits a linear direct relationship with the load device power consumption.

Because of this, in order to amortize this power overhead and achieve power savings, the policy driving the power-gating mechanism must be carefully designed to avoid to switch off devices unnecessarily. In this sense, the *break-even point* (BET) is defined as the minimum amount of cycles a given device must be switched off in order to amortize its switching on power overhead. Therefore, in order to achieve power savings, the power-gating policy must guarantee to prevent to violate the BET.

Power-gating works, similarly to DVFS, also requiring to monitor each device to be power-gated in order to collect status information to be able to take the decision of switching it on/off or not. Similarly, power-gating granularity is an important factor when designing power-gating mechanisms. Some works bet on typical power-gating mechanisms for NoCs in which whole routers are power-gated as in [41], in which authors propose *Router Parking* (*RP*). RP works essentially by powering off routers associated to sleeping cores. They use a centralized controller (*Fabric Manager*) which collects the state of the network, takes the decisions of powering on/off each router and sends this decision to each router. However, usually not all buffers in a given router are equally used and to switch a whole router off decreases significantly the available paths to reach all nodes, even causing some nodes to be unreachable under deterministic routing. Therefore, switching a whole router off might be an overkill which can potentially decrease the NoC performance unnecessarily. To solve this, the RP proposal reroutes traffic around parked routers, which might increase latency and power. Authors propose 3 different *RP* flavors: RP-A (aggressive) which parks as many routers as possible to improve power savings, RP-C (conservative) which carefully selects a small set of routers to be parked, and RP-Adp (adaptive) which selects between RP-A and RP-C dynamically depending on network utilization. This work achieves large power savings but to power whole routers off makes the complexity of this proposal to increase due to the traffic detours and the need to handle corner cases caused by network routing reconfiguration.

As to completely switching routers off incurs in several difficulties due to the loss of connectivity, some works try to avoid this scheme by implementing alternative mechanisms that guarantee minimum services. For instance, authors in [42] propose, instead of switching the whole router off, to decrease the available number of Virtual Channels (VCs) by switching some of them off when the traffic load is low. Other proposals stand by switching off huge parts of the router and enabling bypasses, which are simpler, thereby less power-hungry. An example of this is described in [43], in which NoRD is proposed. NoRD consists in powering routers off but enabling bypasses at powered off routers in order to enable a guaranteed path at each router. This bypass also enables the NIs to inject and eject traffic to/from the network even when its associated router is powered off. The router overhead is low as it needs an inexpensive logic. However, concerning this proposal, the complexity associated to the bypass flow control and VC selection is moved to the NI, which may increase its complexity and power consumption. Similarly, in [44] authors propose TooT, which relies on the fact that most of the traffic

crosses routers making no turns. Based on this fact, TooT switches most parts of the router off, and keeps on only a very reduced version of the router and one latch per port, allowing to forward traffic that requires no turns. However, effectiveness of this sort of strategies depends on the traffic pattern and may not work properly under some circumstances.

Some works bet on fine-grained power-gating approaches to switch unused small parts of the routers, thus, minimizing the loss of connectivity while achieving similar power-savings. As stated in [45], buffers represent one of the most power-hungry parts of a NoC router. Because of this, router buffers represent one of the most common targets when designing fine-grained power-gating techniques. For instance, authors of [46] follow this approach by proposing a buffer power-gating mechanism which makes use of *lookahead routing* to offset the amount of time necessary to powering the buffer on. By using *lookahead routing* each node is able to know in advance the path the message will follow two hops away from it. Each router is connected to the routers located two hops away on each dimension so that the n-th router is able to request powering the buffer on for the *(n+2)*-th router buffers. In this way, the buffers are powered on a few cycles before the first flit arrives to the (n+2)-th router. However, this proposal requires to wire from each router to the one located at two hops on each dimension, which may be expensive. Another interesting proposal is described in [47], in which authors propose *Power Punch*. This proposal consists in sending power-gating signals through the network in order to switch buffers on/off as long as these signals are delivered through buffers. The key of this work is the way in which the authors propose to aggregate in the same message different power-gating signals during their delivery through the network, alleviating the overhead caused by these signals.

Chapter 3

Proposed Techniques

3.1 Congestion Management

In this chapter we describe the different techniques provided in this thesis and their evaluation. For the sake of understanding they are collected from the associated publications exposed in the previous chapters, avoiding in this chapter any duplicity between publications.

In this thesis we propose to combine congestion management mechanisms with power-saving proposals to overcome known performance degradation effects of DVFS-based techniques. To deal with congestion we share the point of view of authors in [15]. In that work, authors do not to deal with congestion itself but with the HoL-blocking effect by identifying congested flows and isolating them into special queues from the non-congested traffic.

In order to combine congestion management mechanisms with power-saving techniques, first we develop congestion management mechanisms following the idea of removing the HoL-blocking by congested traffic isolation. We first introduce BAHIA and then ICARO followed by the evaluations of both.

3.1.1 BAHIA Description

BAHIA (Burst Aware HoL-blocking Injection Avoidance) provides a method to isolate, at runtime, detected bursty traffic in a network, in order to prevent bursts from causing HoL-blocking. Detection of bursty traffic is performed at any end-node receiving a burst. All the end-nodes are then notified of this detection, so that thereafter the bursty traffic can be identified in order to be separated from non-bursty traffic, thereby avoiding the HoL-blocking that the former could produce to the latter. BAHIA makes use of Virtual Networks to separate traffic, hence BAHIA requires at least two virtual networks: the

"regular" virtual network (hereafter regular-VN), for non-bursty traffic, and an "extra" virtual network (hereafter extra-VN)[1] for bursty traffic. Therefore, if no bursty traffic is detected, the traffic is injected always through the regular-VN, but when a burst is detected, the bursty flow is mapped to the extra-VN. It is worth mentioning that, although BAHIA makes a special use of virtual networks, it supports virtual channels implementation over such virtual networks.

3.1.1.1 Burst Detection

As mentioned above, the detection of bursty traffic is performed at the end-node receiving the burst. For that purpose, each end-node periodically calculates its rate of received traffic. The traffic rate is calculated every "polling interval" (PI) cycles, which is a predefined parameter of BAHIA. If that rate exceeds a given high-threshold (HT) value, this end-node will notify the other end-nodes that it is receiving bursty traffic. Similarly, any end-node notifying bursty traffic must be able to detect the end of the burst. For that purpose, the traffic reception rate is compared with a given low-threshold (LT) value. Accordingly, in this case, all the end-nodes will be notified about the end of the burst. It is worth mentioning that an appropriate configuration of the two aforementioned thresholds (upper and lower) is important to achieve the best BAHIA performance. Indeed, in our evaluation experiments we have thoroughly tuned these values, as explained in Section 3.1.3.1.

On the other hand, an alternative burst detection mechanism could be conceived at the sender. Indeed, if a node is going to inject a burst then it can know that in advance. However, detecting at the senders cannot guarantee the effective detection of different lightweight bursts from different sources to the same destination, thereby being not so efficient in preventing the negative effects of aggregate bursts. Hence, we opted for performing burst detection at the receiver part of the end-nodes.

3.1.1.2 Burst Notification

In order to notify a traffic burst to all the end-nodes, BAHIA makes use of a simple dedicated signaling network (Burst Notification Network, BNN). Basically, the whole BNN is a set of one-bit-wide overlapped control networks, each one managed by a specific end-node. Each one-bit control network connects its manager end-node to the rest of end-nodes, the former being the only one able to activate/deactivate the signal (i.e. to set the bit to one/zero) of this control network, while the latter being just signal receivers. Thus, every end-node owns an exclusive one-bit-wide control network to notify bursts events

[1] In some publications composing the papers compendium of this thesis, terms used for the *regular-VN* and the *extra-VN* are *default-VN* and *slow-VN* respectively. However, for the sake of clarity, in this chapter we unified the VNs nomenclature of all proposals using *regular-VN* and *extra-VN* terms.

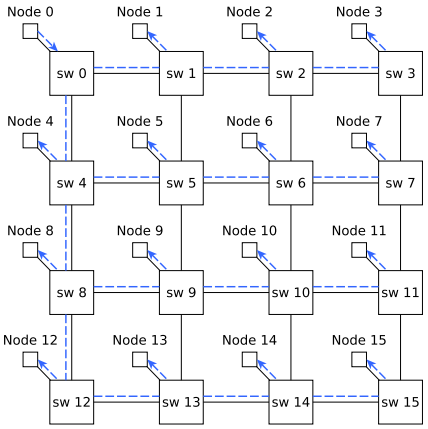


FIGURE 3.1: Example of BNN network for node 0.

to the rest of end-nodes. Figure 3.1 shows the one-bit-wide control network managed by end-node 0, but note that every end-node owns a similar control network, so that in this 4×4 mesh network, there would be other 15 one-bit-wide control networks besides the one shown. The overlapping of these control networks allows every end-node to notify bursts without risk of collisions with notifications from other end-nodes. Therefore, the BNN can be viewed as an N-bit-link, where N is the number of end-nodes and every bit (wire) in the link corresponds to a specific end-node in the system.

Any end-node notifies the rest of end-nodes of a burst by setting to high value the signal of its BNN line. Due to the simplicity of that signal, it reaches all the end-nodes in a few cycles. The time spent in propagating and processing this signal is modeled by the "notification delay" (ND) parameter in the simulations. It is worth mentioning that the processing of this signal is negligible (and so the required hardware). Regarding the area overhead introduced by the BNN, in [48] an additional dual-network for routing data transmission is proposed. This dual-network comprises the logic needed for coding and decoding data, flow controlling, handling transmission failures mechanisms, etc. Despite of its relative complexity, the authors of the referred paper conclude that the area overhead of the hardware required to implement their proposal is 12.5% of the switch. Note that the BNN consists just in a set of wires which are set to a high or low signal value for notifying, hence no logic for processing data is needed. Therefore, relying on the hardware overhead study performed in [48], we can conclude that the hardware overhead for implementing the BNN is negligible.

3.1.1.3 Traffic Separation

Every end-node implements a "burstiness bit-vector", each bit corresponding to a specific end-node in the network, as can be seen in Figure 3.2. When an end-node detects a high signal value in the BNN line associated to an end-node, the former will set to

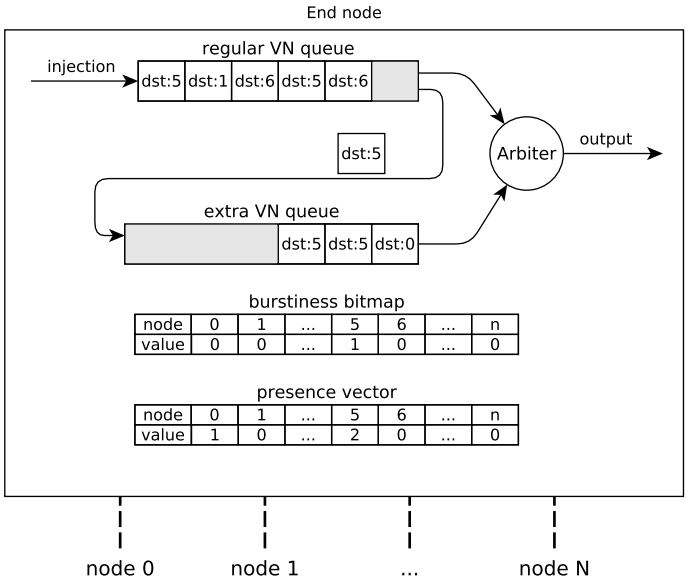


FIGURE 3.2: Burstiness bit vector implemented at each end-node.

one in its burstiness bit-vector the bit corresponding to the latter. Once generated, all the messages are mapped to the regular-VN, thus they are initially stored in the queue associated to that virtual network (regular-VN queue). At every clock cycle the head of all regular-VN queues are checked searching for the head of a message: if one is found, its destination is checked to obtain the value of the corresponding bit in the burstiness bit-vector. If that bit is set to zero, the message remains mapped to the regular-VN; otherwise, the whole message is transferred to the extra-VN. These checking&transferring process can be executed while reading from the queue currently selected for injecting, hence a message at the head of a queue can be transferred to the extra-VN queue while the next one is injected (even when it has to be transferred to the extra-VN too). It is worth mentioning that in a real hardware implementation the burstiness bit-vector could be replaced by simply inspecting the signals in the BNN lines.

In extreme scenarios with several bursts addressed to many end-nodes, and with large burst duration, the extra-VN queues may get full. In these cases messages cannot be transferred from the regular-VN queues to the extra-VN ones, thus the regular-VN queues with messages at their head waiting to be transferred to the extra-VN queue will get blocked until messages at the extra-VN queue are drained. Note that it is an extremely unlikely case: indeed, when simulations were performed with realistic benchmarks this never happened.

Figure 3.2 shows the basic structure of the sender part of an end-node that has messages addressed to end-nodes 1, 5 and 6. The message at the header is addressed to end-node

5. The bit corresponding to end-node 5 in the burstiness bit-vector is set to one (i.e. end-node 5 previously notified that it was receiving a burst), so this message must be mapped to the extra-VN. Indeed, before injection, the arbiter at the sender node checks the burstiness bit-vector and transfers the message addressed to end-node 5 to the extra-VN queue, so that this message will be later injected from that queue.

It is worth pointing out that, although messages are injected either from the regular-VN queue or from the extra one, all of them are initially mapped to the regular-VN. This is because, if an end-node directly maps to the extra-VN the messages addressed to an end-node that has recently notified a burst, there may be messages still stored in the regular-VN queue that are addressed to the same destination, and this could introduce out-of-order message injection (and so delivery) as queue selection policy is based on a simple round-robin algorithm. Hence, to guarantee in-order message injection and delivery, all the messages are first mapped to the regular-VN and the arbiter is provided with some additional intelligence to check the burstiness bit-vector, in order to evaluate whether a message should be directly injected from the regular-VN queue or it should be transferred (by changing pointers) to the extra-VN.

Once an end-node notifies that it is no longer receiving bursty traffic (by setting to low value the signal of its BNN line), the other end-nodes will reset the corresponding bit in their burstiness bit-vector. Thereafter, new messages addressed to this end-node will be injected from the regular-VN queue. However, this may also introduce out-of-order message injection and delivery, as messages addressed to the end-node may remain in the extra-VN queue. The example of Figure 3.2 also shows a situation where there are messages in the extra-VN queue addressed to an end-node (specifically, end-node 0) whose associated bit in the burstiness bit-vector has changed from one to zero.

In these cases, in-order packet delivery can be preserved if messages addressed to a specific end-node are injected from the regular-VN queue only if there are no messages addressed to the same destination in the extra-VN queue; otherwise, the packet must be transferred from the regular-VN queue to the extra-VN one. However, this makes necessary other information than that of the burstiness bit-vector, besides some additional tasks for the arbiter. Specifically, every end-node in BAHIA implements a presence vector that contains an element per end-node in the network, and every element is a counter indicating how many messages addressed to this end-node are stored in the extra-VN queue. Every counter is incremented each time a message addressed to the corresponding end-node is moved from the regular-VN to the extra-VN one, and decremented when messages are injected towards that end-node from the extra-VN. When a message reaches the head of the regular-VN queue and the bit associated with its destination in the burstiness bit-vector is set to zero, the counter associated with that destination in the presence vector is also inspected: if the value of that counter is zero, the message is injected from the regular-VN queue; otherwise, the message is moved to the extra-VN queue.

Note that, although BAHIA has been evaluated in this work assuming a deterministic, dimension-order (XY) routing algorithm, it is suitable to any deterministic or adaptive routing algorithm. However, in-order message delivery is only granted using deterministic routing.

The whole mechanism necessary to keep in-order message injection is a post-processing mechanism, in the sense that messages are mapped to their final virtual network once they reach the head of the regular-VN queue, and not before. As mentioned above, the arbiter should be in charge of performing this post-processing mechanism, that can be summarized in the next pseudocode:

```
for each regular-VN in the end-node do
    if !isVNempty(vn) then
        if isNodeReceivingBurst(msg.destination) ||
          numFlitsInExtraVN(msg.destination) > 0 then
            moveMessageToExtraVN(msg);
        end
    end
end
```

3.1.2 ICARO Description

As mentioned above, BAHIA is designed as a first approach since detecting bursty traffic is simpler than detecting congestion. However, bursty traffic not necessarily may lead in performance degradation or HoL-blocking, thus this approach may not be appropriate for detecting and avoiding harmful traffic. To solve this, we propose ICARO, a more sophisticated and accurate mechanism intended specifically for dealing with congestion at routers.

The purpose of ICARO is not removing congestion but preventing the HoL-blocking caused by congestion. Indeed, ICARO manages to solve this problem by identifying congested flows, then isolating them into special Virtual Networks (VNs) while keeping the non-congested data flows in different regular VNs. By doing this, ICARO separates congested flows from non-congested ones, thus preventing HoL-blocking and so increasing network performance. Similarly to BAHIA, ICARO needs at least two VNs: one *regular-VN* (for non-congested traffic) and one *extra-VN* (for congested traffic). Nevertheless, ICARO may be configured to work with several regular-VNs and also with several extra-VNs. Note that ICARO is a reactive mechanism in the sense that all the system works normally in absence of congestion, keeping the system performance in the same values as the baseline (i.e. the same scenario without ICARO). However, when congestion is detected ICARO reacts to keep network performance by preventing the congestion harmful effects.

ICARO functionality can be divided into three stages: first, congested points in the network are detected at routers; then routers notify the sources of this detection; finally, the sources map the traffic flows either to a extra-VN or to a regular-VN depending on whether or not the injected flow will traverse congested points. These three stages are thoroughly described in the next subsections.

3.1.2.1 Congestion Detection

ICARO is based on detecting *congested points*, defined as output ports persistently oversubscribed. According to the definition of contention given before, ICARO considers that exists contention for an output port if two or more flows request that output port from different input ports. In order to detect whether this contention is persistent, an additional metric is used. This metric consists in counting the number of messages requesting the contended output port. This count is computed per VN at input ports, increasing its value when a new message requesting the output port arrives to the input queue, and decrementing it when the whole message leaves the queue. Every time this count is modified, it is compared with a threshold (SAT_THR) whose value is a configurable parameter of ICARO. Depending on the value of SAT_THR, the congestion-trigger sensitivity of ICARO is lower or higher. As each input port may contain several VNs, an input port is considered as exceeding SAT_THR for an output port if anyone of its VNs does it. Therefore, an output port is considered as a congested point when, in two or more input ports, there are VNs exceeding SAT_THR for that output port.

ICARO must also detect the end of congestion. For this, an hysteresis technique is used. In a few words, once an output port is detected as congested, ICARO detects the end of congestion when the number of messages requesting that output port (in all the VNs) falls below the UNSAT_THR threshold, being UNSAT_THR $<$ SAT_THR. The pseudo-code in Algorithm 1 describes the whole mechanism.

3.1.2.2 Congestion Notification

Once a congested point is detected (or when a previously congested point is no longer congested), sources must be notified in order to isolate (or stop isolating) congested flows into extra-VNs. To deliver this notification ICARO employs a simple dedicated network called CNN (Congestion Notification Network) to send data about the status of the ports to all NIs in the network. The CNN consists in a P-bits-width ring-network to which all NIs and routers are connected, being $P = log_2(NumNodes) + Router\_Radix + 1$. Obviously, in the CNN routers act always as injectors and NIs as receivers so there is no media-access conflicts between NIs, but there may be conflicts between routers. To solve this, this ring is segmented by registers, so that each router has an associated register which separates the signals coming from the previous router from the signals

```
for each output_port do
    for each input_port do
        port_saturated = FALSE;
        for each vn do
            if isVNsaturated(input_port, vn)==TRUE then
                if getNumRequests(vn, output_port) < UNSAT_THR then
                    port_saturated = FALSE;
                    markVNasUNSaturated(input_port, vn);
                else
                    port_saturated = TRUE;
                    break;
                end
            else
                if getNumRequests(vn, output_port) > SAT_THR then
                    port_saturated = TRUE;
                    markVNasSaturated(input_port, vn);
                    break;
                else
                    port_saturated = FALSE;
                end
            end
        end
        if port_saturated==TRUE then
            num_ports_saturated++;
        end
    end
    if num_ports_saturated >= 2 then
        markAsCongested(output_port);
    else
        markAsNoCongested(output_port);
    end
end
```

Algorithm 1: Congestion/no-congestion detection algorithm for ICARO.

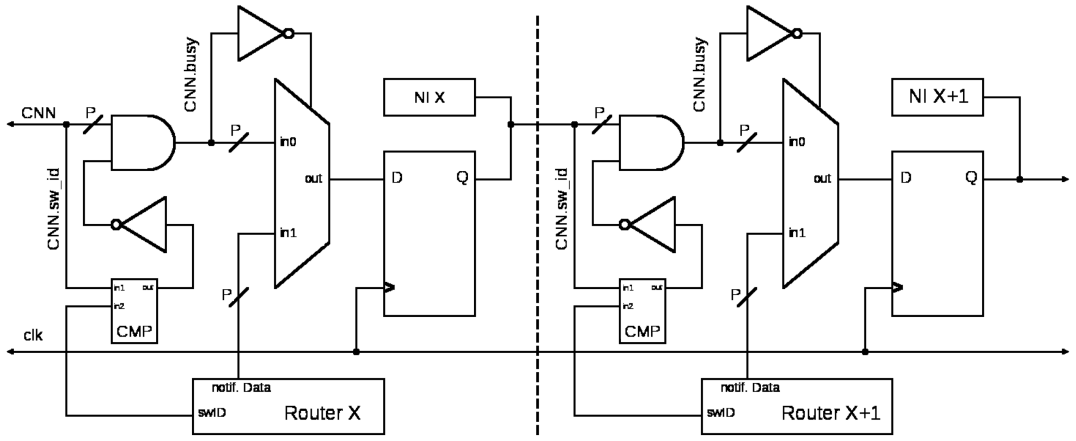


FIGURE 3.3: CNN registers example for ICARO.

being injected from the current router to the next one. The signals at each register are propagated to the next one at each clock cycle. A schematic definition of two consecutive routers is shown in Figure 3.3.

When a router needs to inject data into the network, it waits until its associated register gets free. In practice, the router just keeps the port-status signal at the input of a multiplexer, which selects the register input signal depending on the busy bit sent by the previous register. When the current router register gets free, the multiplexer injects

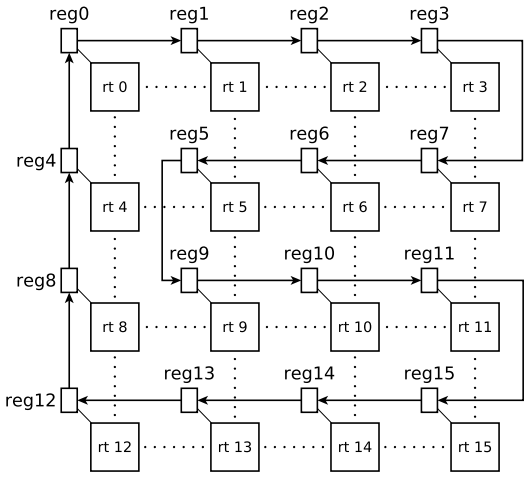


FIGURE 3.4: Complete congestion notification network (CNN) for ICARO.

the signal to the register and such signal is propagated to the next register at the next cycle, and read by the next NI at the same time. When the port status data generated from router x returns to the router x (the data has completed the loop along the ring), such data is dropped and the register x is freed. A complete CNN is shown in Figure 3.4.

The congested-point data sent through the network consists of the router ID coded in binary, a bitmap corresponding to all ports in the router (a bit set to 1 means that the port corresponding to the bit position is congested, otherwise the port is not congested), and an additional bit set to 1 to indicate a valid signal (busy bit). All data is transmitted in parallel, so the CNN must be P-bits-wide.

As some congestion notifications may be dropped at Network Interfaces (NIs) (explained later) or simply lost due to transient failures, the status of the ports is transmitted regularly (re-sync mechanism) to keep the congested-points data coherent at NIs. The frequency at which such data is transmitted is a configurable parameter of ICARO.

Note that this mechanism may not scale for very large systems, as notifications may take too much time to reach all nodes in the network, this delay spoiling the performance improvement achieved by ICARO. Thus, for large systems, instead of using an unidirectional ring to deliver notifications, a hierarchical rings arrangement can be used.

3.1.2.3 Congestion Isolation

Congestion isolation is performed at NIs. As commented in the previous sections, ICARO and BAHIA make use of at least two VNs: one extra-VN and one regular-VN. All flows are always mapped first to regular-VNs, but a module called *post-processor* is in charge of checking the head of all regular-VNs to find messages that should be

re-mapped to extra-VNs. The post-processor checks all queues each cycle: If it finds a head of message, the destination is analyzed in order to check whether or not this message will traverse a *congested point*. If so, the message is re-mapped to a extra-VN. In case of having more than one extra-VN available, the re-mapping module follows a *modulo-mapping* [49] strategy. Since we make use of VNs instead of VCs, messages injected to the network through a given VN are never moved to another VN, as this is the key of isolation mechanism. Messages are provided in their header with a *VN id* prior injection. Such *VN id* is read by routers along the path in order to know in which VN the message must be mapped to. The NI arbiter can be a typical arbiter (such as a *round-robin* arbiter). Note that the re-mapping mechanism can be executed in parallel with the injection of a message from other VNs except the one from which a message is being re-mapped. Also note that ICARO is intended to be used with deadlock-free deterministic routing algorithms (e.g. XY), so no deadlocks can arise.

Contrary to BAHIA, in ICARO NIs must implement a mechanism to manage and store the congested-points data that must be available for the post-processor. This mechanism mainly consists in a cache memory and some additional logic. Figure 3.5 shows a diagram explaining the notification storing process. When a notification arrives to the NI, first this notification is *deserialized*, then traverses some filters (contained in the *Notification-processing* module in Figure 3.5), and is finally stored in the cache. This cache may be implemented as flip-flop registers and it is arranged in several rows and two columns, each row corresponding to a notification while each column corresponds to the data fields contained in the notification (router ID and port). As explained previously, notifications arrive through the CNN as a router identifier coded in binary and a bitmap describing the status of each port. However, not all the port-status data is relevant to the NI since not all ports of a given router are reachable from the receiver NI. Because of this, and also to speed up the cache queries, each notification received through the CNN is split internally into as many notifications as the router radix value (for being able to discard individual port-status notifications). Once the router notification is split into port notifications made of *router, port_status* paired values, each notification is stored in a temporary buffer (*deserializer buffer*). This buffer receives a notification containing the status of all ports of the router at once, allowing the next functional module to read and process each port notification one by one at each cycle. This allows to receive and process properly one notification (containing all ports status) at each clock cycle through the CNN (in extreme cases) during a lapse of time, depending on this temporary buffer size. In case of *deserializer-buffer* overflow, notifications are dropped relying on the *re-sync mechanism* that allows to receive and process them later safely.

Once the notifications are stored in the *deserializer buffer*, each notification traverses a filter which discards unreachable points from this NI, thereby optimizing cache utilization. Congestion notifications which pass the filter are stored in the cache. End-of-congestion notifications which pass the filter trigger a matching-mechanism that removes

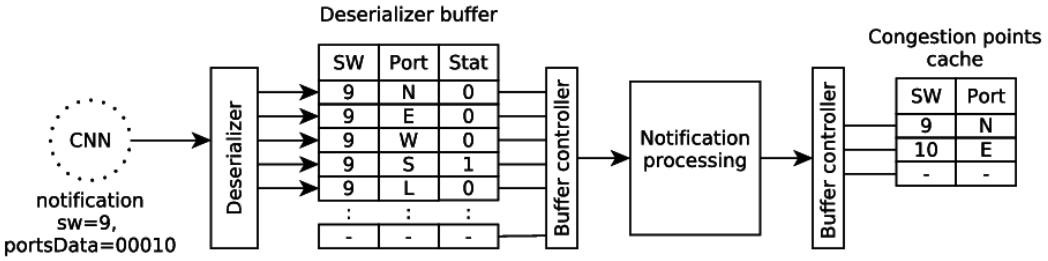


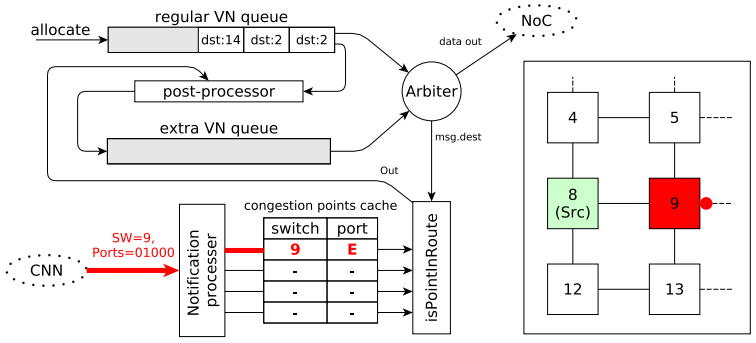
FIGURE 3.5: Notification management.

from the cache congestion notifications which match the same router and port.

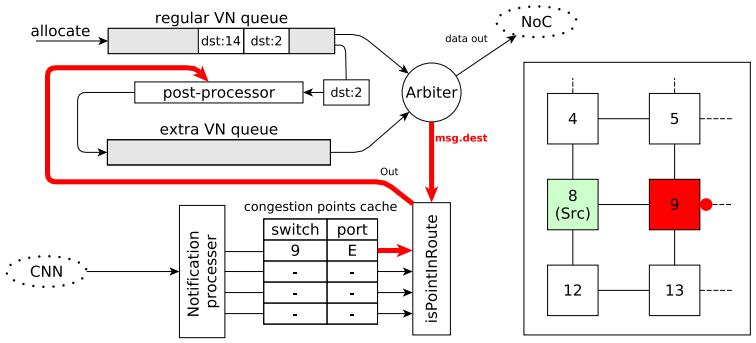
In Figure 3.6 an example of an ICARO NI working in a 4x4 2D mesh is shown. The example shows the behavior of the NI 8 in the network. As can be seen, in Figure 3.6a the NI receives a notification from router 9. This notification contains the router ID and the port status bitmap which informs that the East port of such router is congested while the other ports are in normal state. As shown, the notification is processed and stored in the cache. Next, in Figure 3.6b the message destined to the NI 2 tries to be injected into the network but, as the message will traverse the East port of router 9 (in order to arrive to the NI 2), the message is not injected, instead being re-mapped by the *post-processor* to the *extra-VN*, through which the message will be later injected. Finally, in Figure 3.6c the NI receives a new notification from router 9 informing that none of its ports is congested, so the stored notification is removed, therefore no more messages will be re-mapped to the extra-VN.

Note that, although ICARO makes no use of the extra-VNs in absence of congestion, this does not necessarily lead to lower performance in terms of latency, as the number of VNs only affects latency when network capacity reaches its limit, and ICARO would start using all VNs in this case.

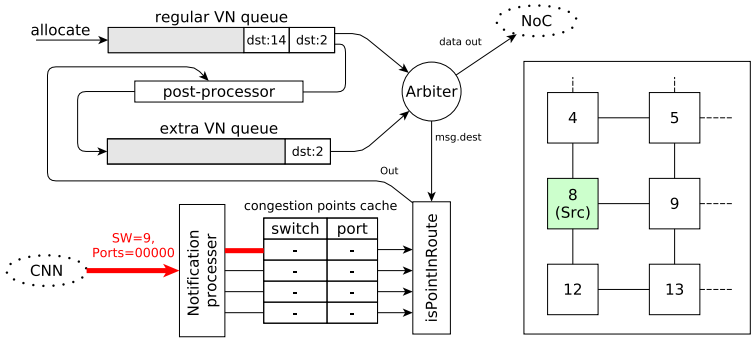
To optimize even more the cache utilization, a congested-points *merge* mechanism can be used. Congestion tends to spread over the network starting from a root point. If congestion spreads towards a given NI, routers contained in the path to the root may notify for congestion sequentially. Due to this, the NI cache may be populated of multiple congested points contained in the same route so all of them but one are redundant since with only the one closer to the NI, the congested route would be covered completely. Thus, an optimization of ICARO consists in a merge mechanism in order to allow the redundant notifications to be discarded in a second filter before storing new congested points. Let us suppose that we have a notification already stored in the cache (notification A). When a new notification arrives (notification B) three circumstances may occur: the new congested point is covered by another, already stored congested point



(A) ICARO receives a congestion notification.



(B) A message is reallocated to the extra-VN.

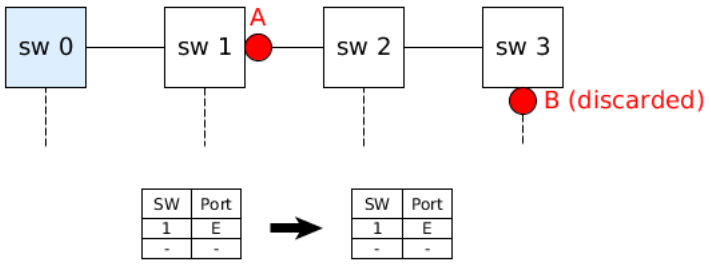


(C) ICARO receives an end-of-congestion notification.

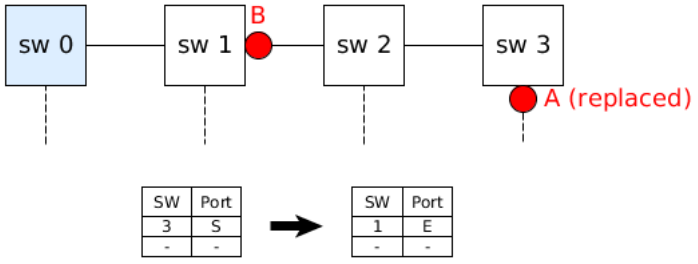
FIGURE 3.6: ICARO NI module mechanism description.

(Figure 3.7a), the new congested point covers one or more already stored congested points (Figure 3.7b), the new congested point belongs to a new route (Figure 3.7c).

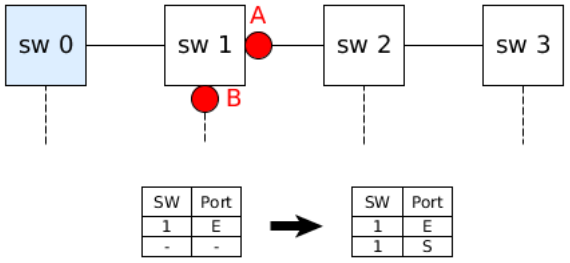
In the first case, the new congested point is useless because all messages crossing B will cross A so the new notification is redundant, therefore can be discarded safely. In the second case, A is contained in B, so A becomes useless if we store B, therefore, A is replaced by B. Also, in such case, more rows may be affected by B. B may cover several already stored congested points, so such congested points can be safely merged, discarding them and storing B instead.



(A) A already covers B, so B is discarded.



(B) B covers A, so A is replaced by B.



(C) A and B belong to separated routes.

FIGURE 3.7: Merge opportunities in ICARO.

However, despite this optimization achieves good results in minimizing the cache utilization, in some scenarios may be counter-productive in performance terms. When using the merge mechanism, the stored congested points tend to get closer to the NI, thereby isolating too much traffic into the extra-VN. Also, since congested points belonging to branches of the congestion tree are usually more volatile than the congestion root, such congested points disappear quickly, thereby removing the congested points at NIs and so becoming the mechanism unstable in some cases. Therefore, this optimization should be used only when the cache size must be critically small due to the lack of silicon area available.

Having presented both, BAHIA and ICARO, we can now identify the main similarities and differences. Both mechanisms rely on the same concept of identifying congestion, rapid notification to end-nodes and separation of traffic (congested from non-congested) using different VNs. Detection in BAHIA is performed at the end-node but in ICARO at every switch output port. The notification infrastructure is different also and more

sophisticated in ICARO with a segmented ring. In BAHIA, notification network is a naive one with just wirings which can have scalability issues. ICARO has caches at each end node identifying congestion points whereas BAHIA has a bit-vector for all end nodes. Both mechanisms use a post-processing mechanism to move messages from regular-VNs to extra-VNs. In summary, ICARO mechanisms is a more advanced mechanism with more complexity in the implementation needed.

In terms of performance, since each mechanism is targeted for different traffic patterns (burst in BAHIA and in-network congestion in ICARO), it is clear that BAHIA will underperform in traffic scenarios correctly handled by ICARO. However, ICARO is also able to address scenarios in which BAHIA performs correctly.

3.1.3 Evaluations

In this section we perform an analysis of performance results obtained with previously described congestion management mechanisms. Since each proposal is focused on dealing with different traffic patterns, both proposals are not really comparable. Therefore, simulation configuration and results for both proposals are separated in their corresponding sections. Following, the first section describes the simulation platform and general configuration parameters used for evaluating both mechanisms. Next, we show results obtained for BAHIA under bursty traffic and finally we show results for ICARO under congested scenarios.

Simulations are performed in a cycle-accurate in-house network-on-chip simulator provided with 4-stage pipelined routers: IB (data storing into the buffer), RT (routing computation), VA/SA (VC allocation/switch allocation, both running in parallel), X (crossbar). Our router model uses wormhole switching with flit-level crossbar and 16-depth queues. Regular 2D mesh topology is used (with XY routing) being each node connected to a single end-node.

3.1.3.1 BAHIA

Results are shown every 5000 simulated cycles (1 transient = 5k cycles). Table 3.1 summarizes the configuration of the scenarios modeled in our experiments. Specifically, the network topology modeled in all the experiments is a 2D mesh built from 16 switches arranged in a 4x4 mesh distribution, each switch being connected to a single end-node. Regarding traffic patterns, for the first analysis an only-synthetic-traffic pattern is used. This pattern (traffic pattern A in Table 3.1) consists of background uniform-traffic injected at a rate of 0.2 flits/cycle/node, together with 2 bursts created by 4 hotspots following a 4-to-1 strategy where nodes inject at 1 flit/cycle/node during 50000 cycles. For the rest of analysis we use traffic pattern B in Table 3.1, which consists of realistic

traffic generated from extrapolated MCSL traces from the H264 video encoder [50], together with background synthetic traffic generated by all nodes injecting at a data rate of 0.3 flits/cycle/node with a uniform distribution of destinations. The latter traffic is used as our goal is to avoid the HoL-blocking that traffic bursts derived from the video encoder processes may cause to other traffic. Regarding the number of virtual networks we consider scenarios with 2, 4, or 8 virtual networks. For the BAHIA case we keep a single extra-VN and the remaining virtual networks are used as normal (regular-VNs) queues, so that all the bursty traffic is mapped into the same virtual network. For the no-BAHIA case all the queues are used equally.

Topology	4x4 2D regular mesh	
Virtual networks	no BAHIA	2, 4 or 8 VNs
	BAHIA	1,3 or 7 regular-VN(s) + 1 extra-VN
Flow control	Stop&go	
Flit size	4 bytes	
Message size	10 flits	
Switch queue size	16 flits	
Traffic patterns	A	uniform 0.2 flits/cycle/node + 4 hotspots 1 flit/cycle/node
	B	h264 video enc. + uniform 0.3 flits/cycle/node

TABLE 3.1: Scenario configuration for bursty traffic in BAHIA.

BAHIA behavior is defined by four parameters: BNN notification delay (ND), high-threshold (HT), low-threshold (LT) and polling interval (PI). In order to explore the robustness of the mechanism, we have carried out an analysis of every parameter by independently simulating different variations of the parameters.

In all the experiments of this analysis traffic pattern B in Table 3.1 has been used, and 2 virtual networks (1 regular-VN + 1 extra-VN) are assumed. The baseline configuration, and the different values of the parameters used for the different simulations are shown in Table 3.2.

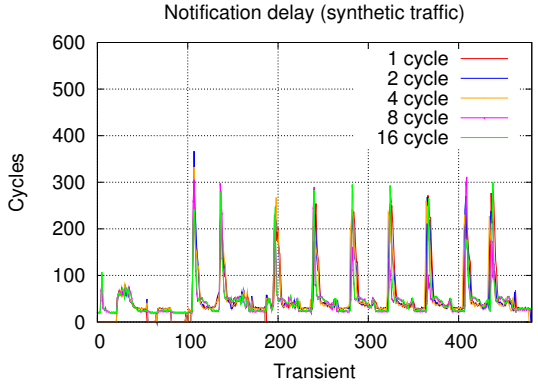


FIGURE 3.8: Notification delay (ND) analysis. Average flit latency.

	HT (flits/cycle)	LT (flits/cycle)	PI (cycles)	ND (cycles)
baseline	0.45	0.35	1000	2
HT analysis	0.4, 0.45, 0.6, 0.75, 0.9	0.35	1000	2
LT analysis	0.45	0.25, 0.35, 0.4	1000	2
PI analysis	0.45	0.35	100, 200, 400, 800, 1600	2
ND analysis	0.45	0.35	1000	1, 2, 4, 8, 16

TABLE 3.2: BAHIA robustness analysis configuration.

First, we analyze the effect when varying the notification delay of the BNN network. In Figure 3.8 we can see the results. This figure shows how notification delay has negligible effects on the BAHIA behavior. The figure shows the latency of messages (only for the synthetic-traffic part of the traffic pattern) when the BAHIA mechanism is running for different notification delays, from 1 cycle up to 16 cycles. As can be seen, latency of messages is unaffected and shows almost the same values. This gives some reliability against unexpected jitter delays and grants flexibility for hardware implementation, since BAHIA has no strict delay requirements.

Next we analyze the effect when varying the value of the detection threshold (high-threshold, HT, at the receiver side of the end-nodes). Figure 3.9 shows results for different values of the HT, ranging from 40% to 90% of traffic reception rate. Note that, the lower the threshold, the more aggressive the mechanism (i.e. more sensitive to traffic bursts), but it may incur in false positives (i.e. non-bursty traffic detected as bursty). By contrast, the higher the threshold, the more selective the mechanism, thus it may not detect lightweight bursty traffic. As we can see in Figure 3.9, all the considered values of HT except the lowest one (40%) produce similar results. Thus we can conclude that this parameter should be configured so that BAHIA is not too sensitive, i.e. the HT value is high enough to avoid false positives.

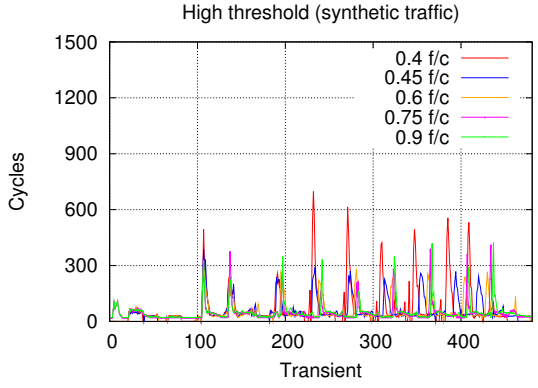


FIGURE 3.9: High-threshold (HT) analysis. Average flit latency.

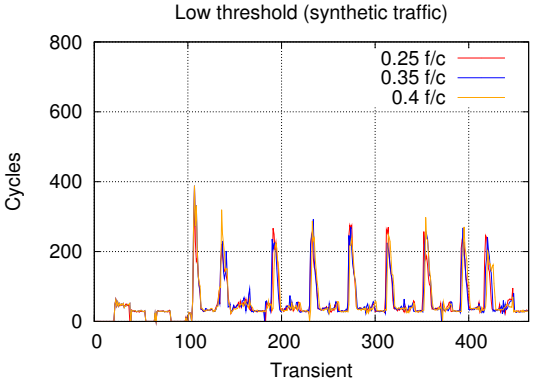


FIGURE 3.10: Low-threshold (LT) analysis. Average flit latency.

Similarly, Figure 3.10 shows the results obtained when varying the value of the low-threshold (used to detect the end of bursts). As can be seen, this parameter does not affect BAHIA performance because, when a burst ends, all virtual networks are free from HoL-blocking effects, so traffic flows smoothly through all virtual networks, thus no matter which virtual network messages are mapped to.

Finally, the value of the polling interval has been tested. Note that the polling interval tuning presents a close relationship with the high-threshold analysis: having a small polling interval may be similar to having a lower HT value. The contrary also applies: a large polling interval could lead to the mechanism filtering short transient bursts. As can be seen in Figure 3.11, for all the considered values we get a similar behavior. We conclude, then, that the smallest polling interval among those considered is convenient and not harmful.

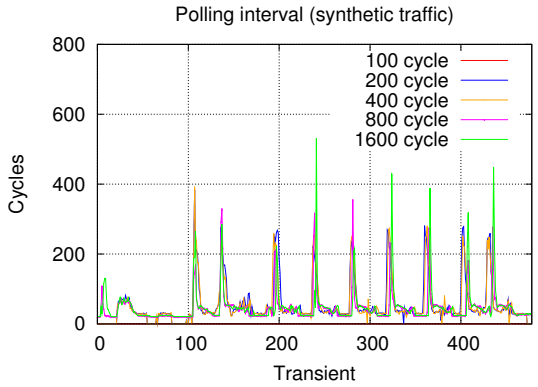


FIGURE 3.11: Polling interval (PI) analysis. Average flit latency.

Summing up, any combination of the considered values for these parameters, except the lowest value for HT, would produce similar results. Thus, hereafter we assume that an appropriate configuration of parameters can be ND=4 cycles, HT=0.6 flits/cycle, LT=0.4 flits/cycle, PI=400 cycles, hence such configuration will be used for the next analysis.

Next, we carry out a general analysis of BAHIA in comparison with similar scenarios without BAHIA in order to quantify the capability of BAHIA to separate bursty traffic from non-bursty one. In the previous section we have delimited a set of appropriate values for the parameters that define BAHIA behavior for the scenario used in the simulations, so in this analysis we have set BAHIA parameters according to these values.

For the next analysis, we have performed simulations for the simplest configuration in terms of virtual networks, i.e. 2 virtual networks (as BAHIA requires a minimum of 1 regular-VN and 1 extra-VN). Figure 3.12 shows the latency achieved with and without BAHIA for traffic pattern B in Table 3.1. Note that the results for the synthetic and MCSL components of the traffic pattern are shown separately, as different series in the same graph. Average numeric results can be seen in the first two entries of Table 3.3. As

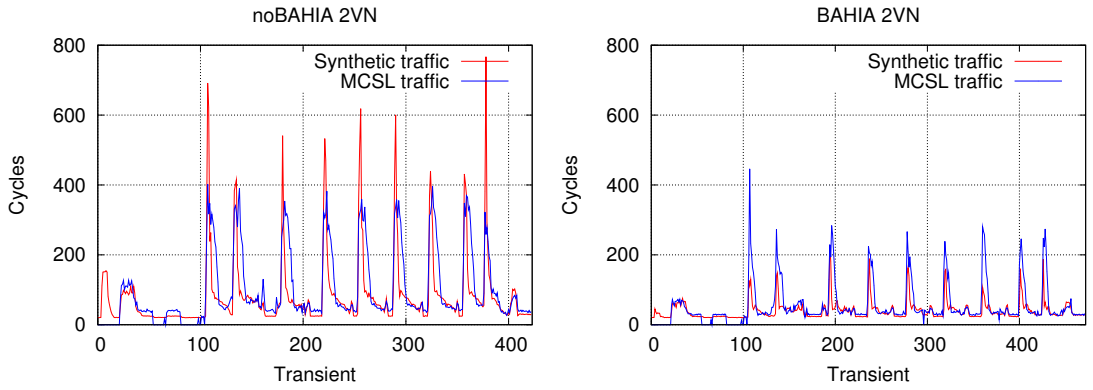


FIGURE 3.12: Latency for BAHIA and no-BAHIA, 2 VNs, traffic pattern B.

Average latency (cycles)			
	noBAHIA	BAHIA	Improvement
2VN synthetic traffic	56.78	40.40	40.54%
2VN MCSL traffic	130.02	121.89	6.67%
4VN synthetic traffic	85.89	39.31	118.47%
4VN MCSL traffic	240.10	102.40	134.46%
8VN synthetic traffic	133.71	37.65	255.15%
8VN MCSL traffic	464.21	110.88	318.66%

TABLE 3.3: Average latency for BAHIA and no-BAHIA scenarios.

can be seen in Figure 3.12 BAHIA achieves a moderated improvement of approximately 40% in the average latency with respect to the no-BAHIA case (seen also in Table 3.3)

Now, we turn our attention to scenarios where different numbers of virtual networks are available. Specifically, we consider the cases of 2, 4, and 8 virtual networks. As mentioned above, for the BAHIA case we keep a single extra-VN, the remaining virtual networks being used for non-bursty traffic. For the no-BAHIA case all the queues are used equally. For this analysis we have used traffic patterns A and B in Table 3.1, in order to get a more complete comparison between BAHIA and no-BAHIA cases.

We firstly analyze results when traffic pattern A in Table 3.1 (i.e. only synthetic traffic) is used. Assuming this traffic pattern, Figure 3.13 shows results of latency along time for a network without BAHIA. The results correspond to the overall average latency for all VNs. Clearly we can see a latency increase when contention exists due to the appearance of hotspots, regardless the number of virtual networks. In Figure 3.14 the overall average latency for all regular-VNs with BAHIA is shown. It can be seen that when traffic bursts start, BAHIA quickly detects an excessive received data rate, thus it is notified to the rest of end-nodes and such traffic is isolated in the extra-VNs, keeping the regular-VNs latency low. On the other hand, as can be seen in Figure 3.15 the latency of the extra-VNs increases as bursts are mapped to them. Note that these

BAHIA results do not significantly vary with the number of virtual networks, and they show clearly how BAHIA isolates traffic-bursts into the extra-VN.

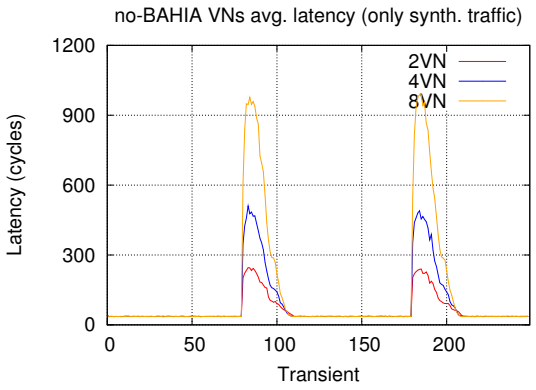


FIGURE 3.13: Overall average latency without BAHIA, traffic pattern A.

Secondly we focus on the results obtained for traffic pattern B in Table 3.1 (i.e. realistic MCSL traces together with synthetic background traffic), with and without BAHIA. Note that the results corresponding to the synthetic part of traffic pattern B are shown separately (in Figure 3.16) from those corresponding to the realistic part (MCSL traces) of this pattern, that are shown in Figure 3.17. Average numeric results can be seen in Table 3.3. As can be seen in these figures, the higher the number of virtual networks, the higher the latency increase in the no-BAHIA case. By contrast, when using BAHIA, non-bursty traffic latency keeps bounded and minimized, reaching high latency reductions up to a factor of 3x.

In order to better appreciate the average latency improvement of BAHIA in relation to the no-BAHIA scenarios with different number of VNs in Figure 3.18 we can see the average latency for such cases. Clearly, without BAHIA latency increments linearly with the number of VNs while the scenarios implementing BAHIA keep it constant.

The BAHIA mechanism is unable to handle efficiently in-network congestion points. Indeed, congestion is detected only at end-points. Nevertheless, BAHIA allows us to identify the potential of dynamically separate flows to avoid congestion. Indeed, BAHIA is used to further deploy the mechanism enabling in-network congestion detection. This is the ICARO mechanism which is described next.

3.1.3.2 ICARO

This section is organized as follows. ICARO results are compared with FVADA and AVADA [25][2]. First, the scenarios used for the simulations are described. Next, ICARO

[2]For a description of FVADA and AVADA, please refer to Section 5.3 of Chapter 5

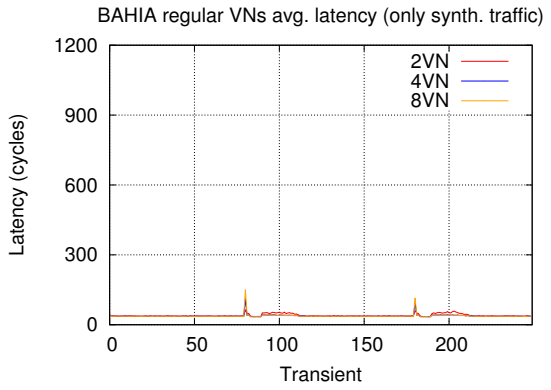


FIGURE 3.14: Regular-VNs average latency with BAHIA, traffic pattern A.

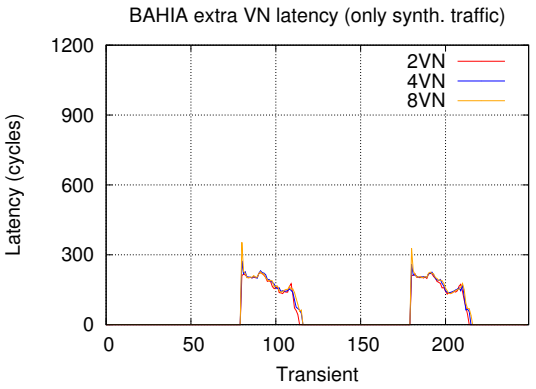


FIGURE 3.15: Extra-VN average latency with BAHIA, traffic pattern A.

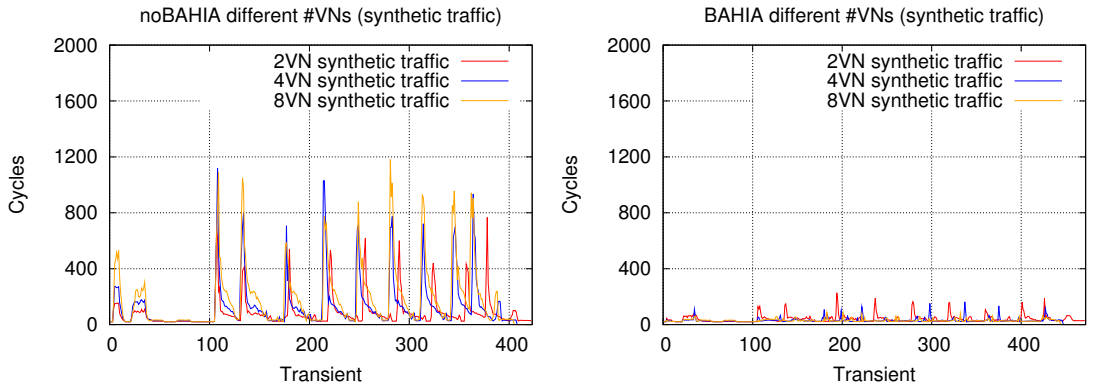


FIGURE 3.16: Latency for the synthetic part of traffic pattern B, without and with BAHIA.

is evaluated in different scenarios varying critical parameters that may affect its performance. Then, a performance analysis is carried out comparing ICARO with FVADA and AVADA and, finally, we perform a physical implementation analysis in order to evaluate the costs in terms of area and power when implementing ICARO.

Simulations configuration parameters not mentioned in this section are set to the same values as for BAHIA simulations. The amount of queues varies depending on the strategy used, e.g. if the amount of queues is 4. A regular 8x8 2D mesh network is used, therefore router radix=5. It is noteworthy that, both FVADA and AVADA make use of virtual channels (VCs) instead of VNs as ICARO does. So, for the baseline scenario we decided to make use of VCs as well. Messages are 5-flits long. Regarding traffic patterns, two types of synthetic traffic patterns are used. On one hand, typical synthetic traffic patterns are used like uniform, tornado, bit-reversal, etc. On the other hand, similarly to BAHIA, as ICARO is a proposal intended to deal with irregular, bursty, hotspot-prone traffic patterns, we drew up a combined traffic pattern. This traffic pattern is composed of a light uniform background traffic and a hotspot component. Hotspots are active only

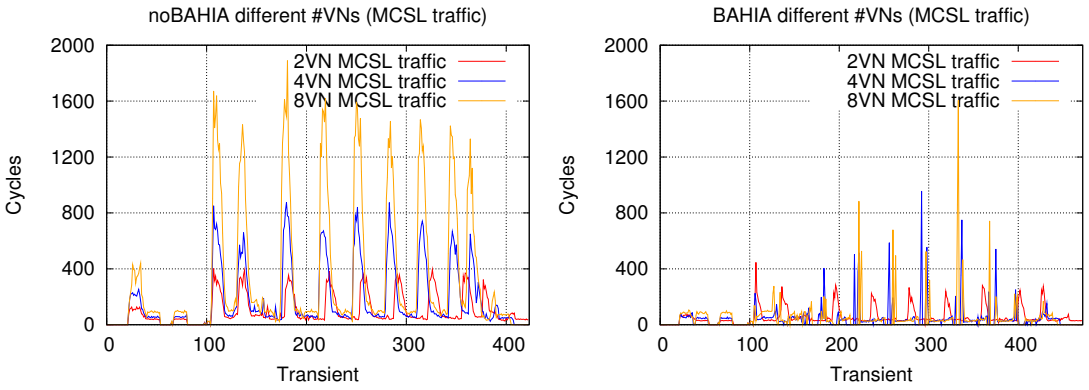


FIGURE 3.17: Latency for the MCSL part of traffic pattern B, without and with BAHIA.

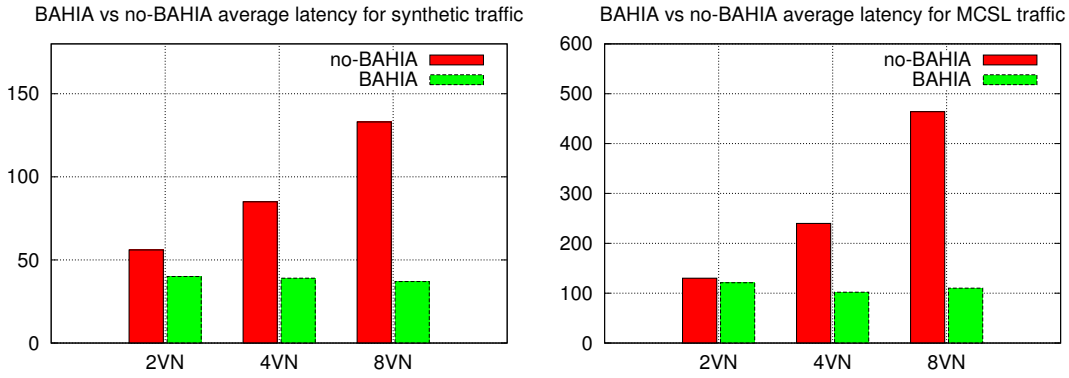


FIGURE 3.18: Average latency without and with BAHIA for the synthetic and MCSL parts of traffic pattern B.

from cycle 10k to 20k. In this way, we have a background traffic which generates no congestion, and another component of aggressive traffic which causes congestion, causing HoL-blocking to the background traffic. Since FVADA/AVADA requires credit-based flow control, all simulations implement it.

In previous sections we stated that ICARO needs at least 2 VNs, one for regular traffic and one for congested traffic. However, the amount of VNs can be increased as much as we need, arranging the VNs in several configurations: 1+1VNs, 2+2VNs, 4+1VNs, 4+4VNs, etc. Also, for ICARO, the cache size is a critical parameter depending on the traffic pattern and network size. So, for the purpose of evaluating the impact of such variables, an analysis is performed. For this analysis the combined hotspot traffic patterns have been used with a background traffic of 0.3 flits/cycle/node.

The evaluation is performed with different VN configurations. In order to graph the network latency for the different configurations, each VNs arrangement has a number assigned that identifies such configuration. Identifier XY is for a configuration with X regular-VNs and Y extra-VNs. We will play with configurations 11, 22, 31, 44 and 71. For the ICARO notifications we assume a propagation delay of 2 cycles for each

hop. Regarding the baseline configuration we use exactly the same configuration as the ICARO scenario with the same number of VNs (considering both VN types: regular and extra-VNs).

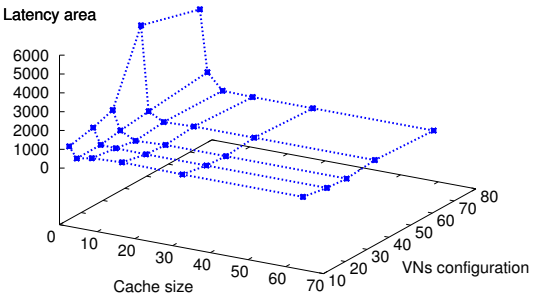
Regarding SAT_THR and UNSAT_THR thresholds used in the congestion-detection mechanism (see Section 3.1.2.1), we performed simulations using different values for these thresholds, in order to evaluate their impact in ICARO and to obtain the optimal values. From the results obtained we conclude that the thresholds values do not have a significant impact on the ICARO behavior while they are confined in a reasonable range. Nevertheless, the best results are achieved for SAT_THR=4 and UNSAT_THR=2, so these values are assumed in the current analysis.

ICARO can work with the re-sync mechanism and/or the merge mechanism. Nevertheless, both mechanisms may cause counter-productive effects, so this analysis has been performed with three combinations of these mechanisms: *no re-sync/no merge, re-sync/no-merge* and *re-sync/merge*. The combination *no re-sync/merge* has not been considered as the purpose of the *merge* mechanism is to save cache slots in scenarios with a high number of notifications due to the re-sync mechanism, therefore it makes no sense to implement the *merge* mechanism when the *re-sync* one is not enabled.

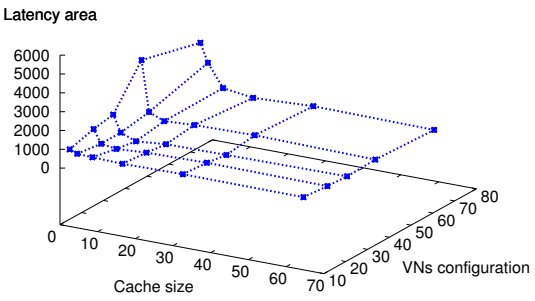
Regarding the *congested points cache*, the following sizes have been used: 2, 4, 8, 16, 32 and 360 (theoretical limit due to the maximum possible congested points that can be given in a 8x8 network). In the graphs, the 360 value has been replaced by 64 value for better viewing. For the *deserializer buffer* size we adopted the policy of using $2 * cache\_size$ entries (determined empirically).

In Figure 3.19 the results are shown. The metric used for measuring the performance is the *latency area*, which consists of the sum of the latency overhead during the whole simulation. The latency overhead is measured as the difference of latency values between the case with congestion present and the case with only background traffic running.

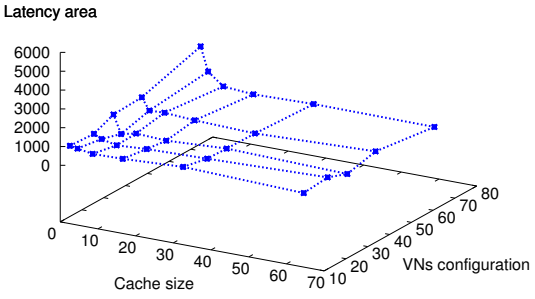
As can be seen, in all cases, with 4 or more cache slots, there is no latency area increase as ICARO has room enough to store all relevant congested points, thus it is able to isolate all the congested traffic properly. However, as the number of cache slots available falls below 4, ICARO is not able to isolate congestion properly. However, as can be seen in Figure 3.19b, the re-sync mechanism helps to alleviate the shortage of cache entries. This is because congested points dropped due to the lack of room are re-notified periodically, so they have more opportunities to be stored. Besides, if we add the merge mechanism (Figure 3.19c), the latency falls even more as the congestion slots are better managed, so that there are more free slots to store congested points. However, the conclusion from this analysis is that with at least 4 cache entries HoL-blocking is completely removed regardless of VN configuration.



(A) No re-sync, no merge.



(B) Re-sync, no merge.



(C) Re-sync and merge.

FIGURE 3.19: ICARO configuration analysis.

Next, the ICARO performance is compared against AVADA and FVADA. First, all techniques are simulated using common traffic patterns. As can be seen in Figure 3.20, for most of the common traffic patterns ICARO keeps the results in similar values to the baseline and the other techniques. In the case of ICARO there is a slight overhead close to saturation. This is due to the fact that it employs VNs instead of VCs. VCs gives more flexibility at the arbitration stage so is expected to perform better than using VNs. However, our proposal goal is not to improve the performance over static traffic patterns but with the combined hotspot one[3].

In Figure 3.21 we can see the latency results for the different mechanisms over combined hotspot traffic pattern. Note that all mechanisms but ICARO use VCs while ICARO uses VNs. In Table 3.4 the ICARO parameters configuration is shown. Let us recall that ICARO aim is to isolate harmful traffic into the extra-VN in order to avoid non-harmful traffic to be affected by the former. To better appreciate the ICARO behavior, latency

[3]In the case of FVADA and AVADA, despite of reproducing exactly the same scenarios the authors used in their evaluations, we could not obtain the results exposed by them for common synthetic traffic patterns.

Parameter	Value
VNs config.	1+1, 3+1 and 7+1
SAT_THR	4
UNSAT_THR	2
Cache size	4
Deserializer buffer size	8
Re-sync	No
Merge	No

TABLE 3.4: ICARO configuration.

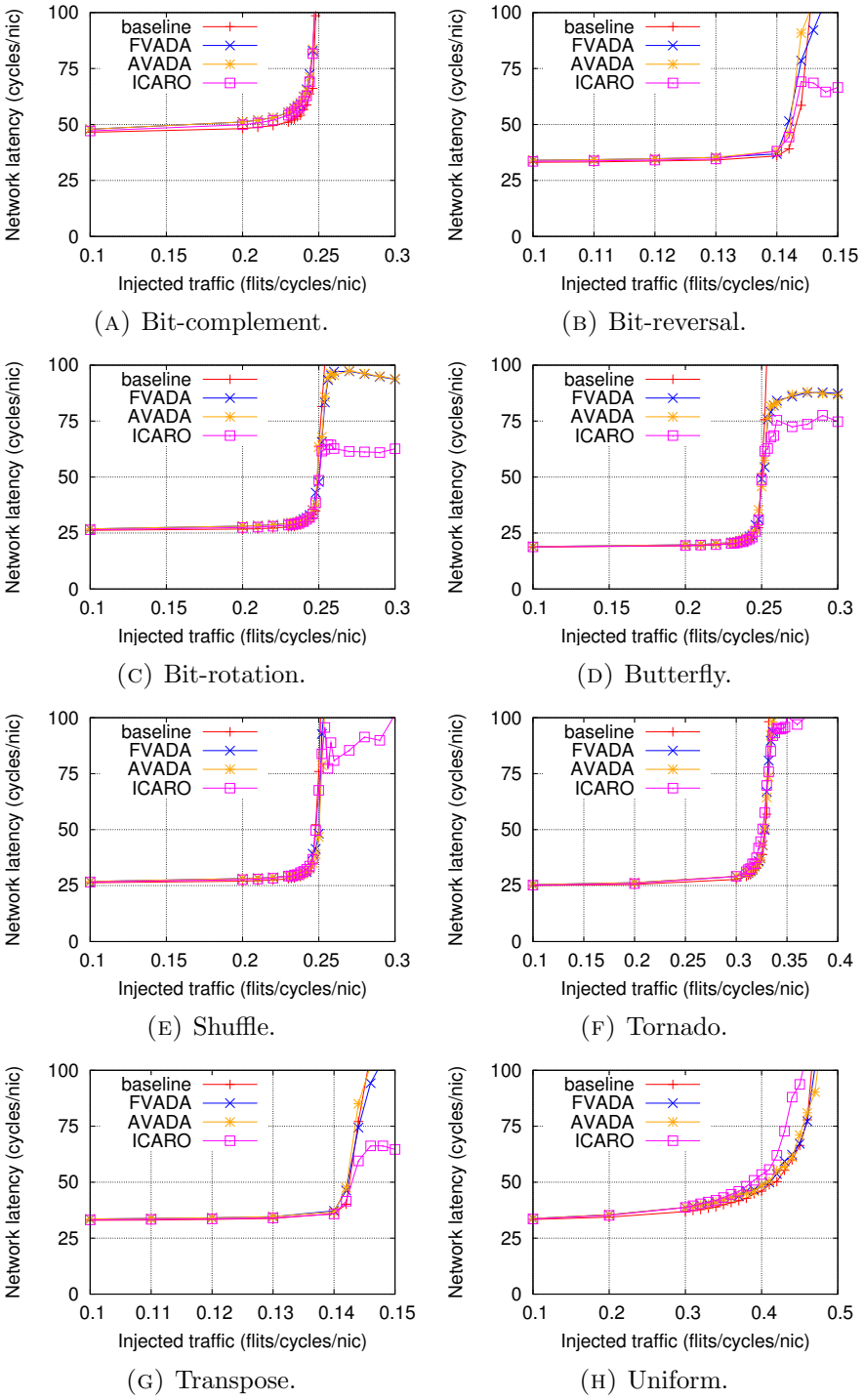
results for our proposal are shown in two graphs: one for network latency average of all regular-VNs and another one for network latency of the extra-VN.

As can be seen, ICARO outperforms all other mechanisms achieving an improvement of up to 82% for the 8VN configuration. In the case of the 2VNs and 8VNs simulation FVADA is not shown because FVADA requires exactly *r-1* VCs (r=router radix). Notice that congestion injection lasts from 10k-cycle to 20k-cycle. The HoL-blocking effects in ICARO are minimized and removed after the congestion builds. However, for the other configurations, congestion remains beyond the 20k-cycle point. They recover performance point only beyond 60k-cycle.

Costs Analysis

Following, the area and power overhead of ICARO is analyzed. To perform this, the ICARO mechanism has been implemented in Verilog using a canonical NI and a wormhole router, both with support for 4 VNs. The router queues have a 4-flit size with a 128-bit flit size. For the NIs the queues have a size of 8 flits. Regarding the ICARO configuration, it has been implemented with support for merge and re-sync mechanisms with a cache size of 4 slots and a deserializer buffer size of 8 slots. To synthesize the Verilog designs, Design Vision tool from Synopsys with 45nm Nangate open cell library [51] (typical conditional) has been used. Then, we performed the place&route process with Encounter tool (from Cadence) to estimate accurately the area overhead. Figure 3.22 shows the results for the area and power overhead of a NI implementing ICARO compared with the baseline NI for different network sizes. In Figure 3.23, the area and power overhead results of our proposal for the router are shown.

ICARO needs additional hardware in order to implement the CNN. However, this hardware is not strictly located either at the router or the NI. This hardware consists of wires interconnecting nodes and the logic associated to these wires (shown in Figure 3.3). In order to fairly evaluate all the hardware overhead imposed by ICARO, the logic associated to the CNN is included in the router overhead. Wires are not taken into account as they do not actually impose area overhead. Indeed, such wires use metalization layers. However, in the design floor-plan, little empty gaps always exist between all tiles to



(A) Bit-complement. (B) Bit-reversal.

(C) Bit-rotation. (D) Butterfly.

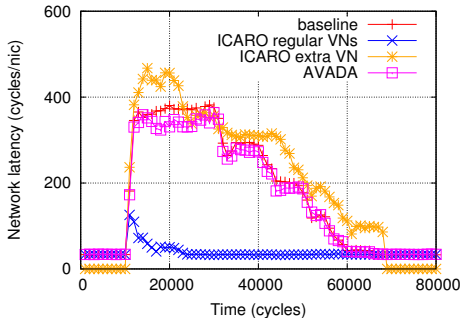
(E) Shuffle. (F) Tornado.

(G) Transpose. (H) Uniform.

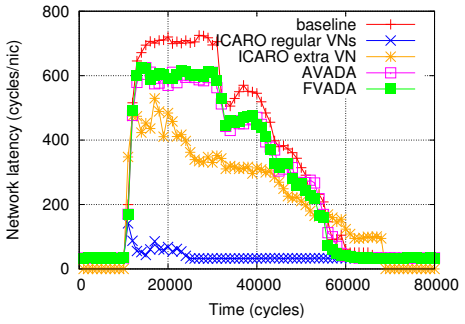
FIGURE 3.20: Typical synthetic traffic patterns.

physically isolate each tile from its neighbors. Provided that these gaps are big enough to physically place all links between tiles, the area spent by the whole CMP remains the same even including the CNN wires.

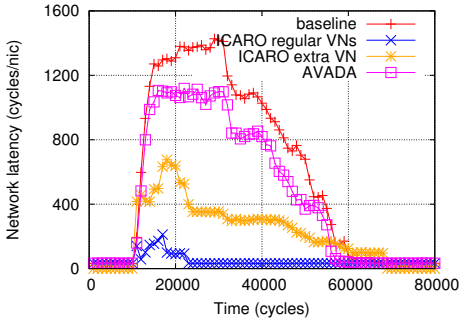
As can be seen in Figure 3.22, for all cases, the area overhead for the NI varies between 3.8% for a 16-node network, and 6% for a 1024-nodes network. For the power overhead



(A) 2VC/2VN.

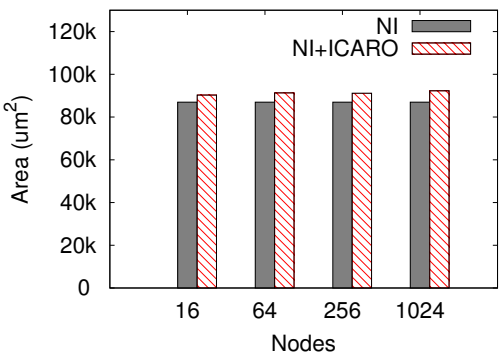


(B) 4VC/4VN.



(C) 8VC/8VN.

FIGURE 3.21: Performance evaluation with hotspot traffic pattern.



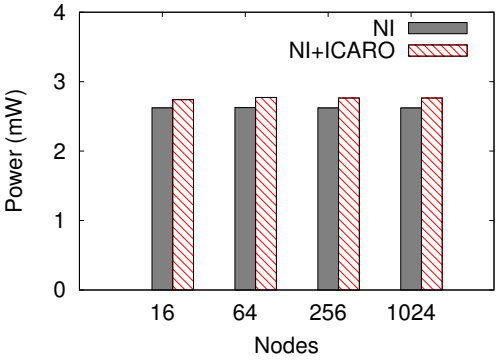


FIGURE 3.22: NI area and power overhead for ICARO.

it varies from 4.5% to 5.4%. In Figure 3.23 the overhead results for the router are shown. For the area, ICARO has an overhead of 6.7% for all cases. In the case of power, values from 6% to 10% have been obtained.

As shown in such results, ICARO demonstrates an acceptable area and power overhead either for the NIs or the routers. In addition, taking into account the results for different networks sizes, seems clear that ICARO scales with the network size with no significant extra area or power overhead.

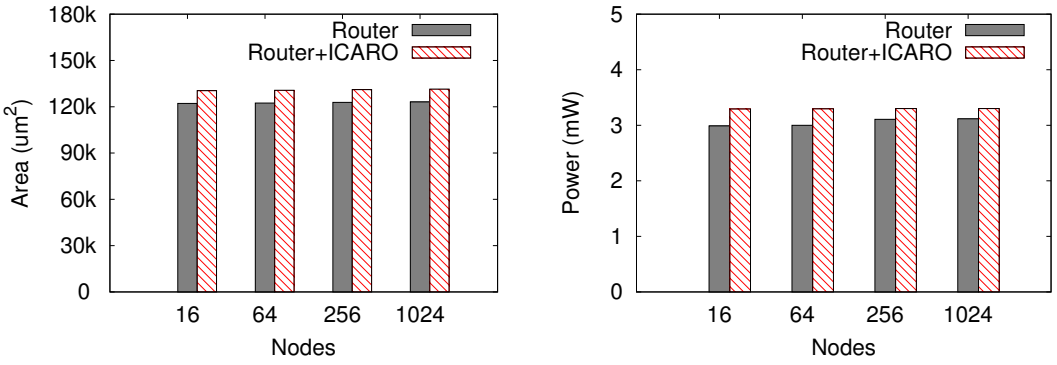


FIGURE 3.23: Router area and power overhead for ICARO.

3.2 Improving DVFS Through Congestion Management

In this section we describe two different strategies to combine ICARO with DVFS. First we propose to combine ICARO with a vanilla version of DVFS through three different strategies to improve power-savings, performance or both. Then, we describe a proposal which combines a latency-driven DVFS together with ICARO, in order to improve its behavior under congested traffic.

3.2.1 ICARO-DVFS

In this section, we present our proposal of combining ICARO with DVFS. As stated previously, the aim of this proposal is to integrate both mechanisms in different ways to improve performance, power or a combination of both parameters. First we describe our DVFS implementation to better understand our proposal and then we describe each integration proposal.

3.2.1.1 Dynamic Voltage and Frequency Scaling

Our DVFS implementation changes voltage and frequency in levels, depending on the performance demand. Each level corresponds to a given pair of voltage-frequency fixed values (Table 3.5). Off-chip VRs are assumed.

DVFS level	$Voltage(V)$	$Freq(GHz)$	DVFS level	$Voltage(V)$	$Freq(GHz)$
Level 1	1.30	3.074	Level 4	1.15	2.281
Level 2	1.25	2.852	Level 5	1.10	1.932
Level 3	1.20	2.588	Level 6	1.05	1.540

TABLE 3.5: DVFS levels assumed in the ICARO-DVFS mechanism (obtained from [1])

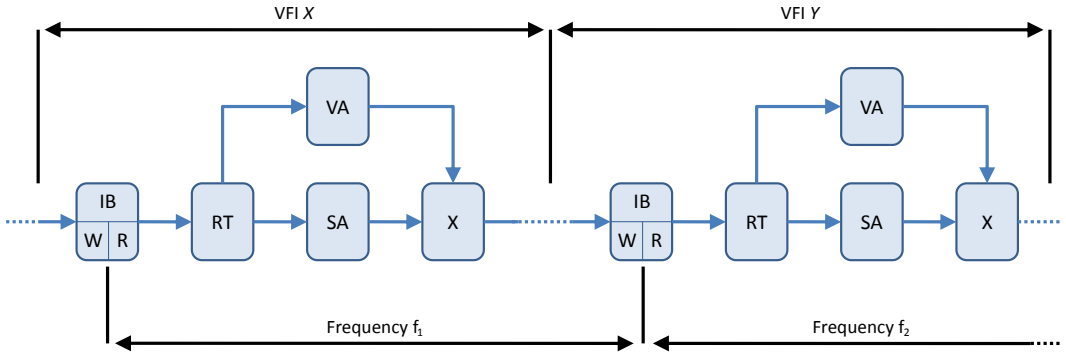


FIGURE 3.24: Two consecutive routers belonging to different VFIs (at the boundary delimiting such VFIs).

DVFS levels are changed by monitoring the occupancy at router buffers. Every *poll_period* cycles across all the monitored routers sample and report their occupancy level. If any buffer exceeds a Q_h threshold the DVFS level is decremented (voltage and frequency are incremented). Accordingly, if all buffers exhibit an occupancy below the Q_l threshold ($Q_h > Q_l$), the level is incremented (frequency and voltage are decremented).

3.2.1.2 Voltage and Frequency Islands

Our DVFS implementation supports VFIs, so voltage and frequency changes can be applied per VFI domain. Each VFI has its own VR which collects metrics from the routers in its domain and changes voltage-frequency accordingly. However, routers at the boundaries between VFIs must be carefully designed as data flows will cross different domains. To address this issue, we implement mixed-clock/mixed-voltage buffers[52][53] which enables to write and read at different frequencies.

Figure 3.24 shows the router implemented at a VFI boundary (see Section 3.1.3 for more information about routers architecture design). The *input buffer* stage is divided into two *sub-stages*, the write part belongs to the frequency domain of the upstream router while the read stage belongs to the current router frequency domain. In this way, the rest of pipeline stages (after IB) are able to work normally at its corresponding frequency. The same is performed for the flow control logic. As credit-based flow control is used, the credits buffer at the downstream router works at both frequencies, at the upstream router frequency for writing and at the current router frequency for reading.

```
if CNNreg.busy && idBelongsToVFI(CNNreg.routerID, thisVFI) then
    if CNNreg.FreqIncr then
        freqIncrVector[routerID] = 1;
        if |freqIncrVector then
            increaseFrequency(thisVFI);
    else if CNNreg.FreqDecr then
        freqDecrVector[routerID] = 1;
        if |freqDecrVector && ~ |freqIncrVector then
            decreaseFrequency(thisVFI);
    else
        Do nothing
    end
end
```

Algorithm 2: DVFS level change algorithm for ICARO-DVFS.

3.2.1.3 Merging ICARO with DVFS

Now we show how ICARO is coupled with DVFS. Basically, ICARO notifies congestion events through the CNN network. This network is extended to send DVFS notifications as well[4]. This is easily achieved by changing the ICARO controller implemented in the router. In addition to detect congestion events, the new logic monitors all input ports queue occupancy and sends two new events through the CNN. Figure 3.25 shows the new notification format. Two bits are added just indicating the router requests for a level increment or decrement.

Buffer occupancy is analyzed in each router and compared against Q_h and Q_l thresholds. If any of the queues exceeds the Q_h threshold the VFI_{inc} bit is set. Once all queues occupancy are below Q_l the VFI_{dec} bit is set. Only when any of those two bits change a DVFS notification is sent through the CNN network.

At each VFI domain, the VFI module reads the notification commands from the CNN network, keeping record of all routers VFI_{inc} and VFI_{dec} bits from its domain. Two n-length bit vectors (being n the number of routers in the VFI domain) are implemented. VFI frequency/voltage is incremented when any of the routers request for such increment (even if a router is requesting for a decrement). Frequency decrement is performed when any router is requesting a decrement and none of the routers are requesting an increment. Figure 3.26 shows the logic, whereas Algorithm 2 shows the algorithm change DVFS levels.

[4]In a typical DVFS implementation, a dedicated logic collects metrics from the network and deliver them to the logic that implements the VFI policy and drives its VR to carry this out. We take advantage of the CNN network to simplify this process.

3.2.1.4 Different ICARO-DVFS Alternatives

Besides the CNN extension, ICARO deals with different virtual networks (VNs) to decouple congested traffic from non-congested traffic. In the minimal implementation (2 VNs), just one VN is used to map congested traffic, the other one remains for non-congested traffic. As commented previously, to couple correctly ICARO and DVFS we need to sense occupancy of input ports queues. However, as we have differentiated VNs we have different options.

As a first alternative, we can sense all input ports queues, including both non-congested traffic VN and congested traffic VN. In this case, DVFS will raise frequencies and voltages whenever traffic increases to levels where congestion appears in the VFI domain. This alternative is referenced to as *ICARO-2VN*.

Another alternative is to raise frequencies and voltages only whenever the non-congested VN congests as well. In this case, the DVFS strategy will raise only when severe congestion appears in the network. In other words, when congestion caused by hotspot traffic affects background traffic in such a way in which causes regular-VNs to exceed Q_h threshold. It is supposed that this alternative will lead to more power-saving results. This alternative will be referenced to as ICARO-1VN.

Finally, a different alternative is to sense all input port's queue, but differently from *ICARO-2VN*, the DVFS strategy will be bounded to a more conservative frequency level. In this case, the maximum frequency will be set to ∼2GHz (instead of ∼3GHz), corresponding to *Level 5* frequency on Table 3.5. The reasoning behind this strategy is the effect ICARO has on performance as will decouple congested traffic from non-congested one. Therefore, increasing frequency for performance reasons will become less critical. This alternative will be referenced to as *ICARO-2GHz*. In addition, this approach allows to simplify VRs by reducing the number of voltage-frequency levels provided. Notice that area consumed by VRs depends directly on the voltage-frequency levels provided.

The three strategies will be analyzed in Section 3.2.4.1. All of them will be compared against three different strategies. The two first strategies will not use DVFS at all and will set the network both to minimum and maximum frequencies. They will be referenced to as *minFreq* and *maxFreq*, respectively and will allow us to set the low and up limits in terms of performance and power. The third strategy will be compounded of DVFS with the defined levels shown in Table 3.5 and sensing the VFI occupancy queues regardless of the congestion effects. This strategy will be referenced to as DVFS in the plots.

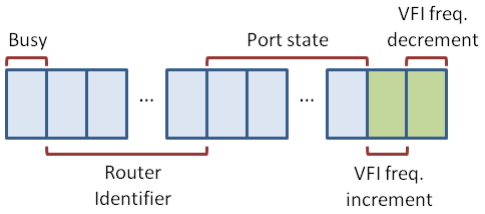


FIGURE 3.25: CNN signal format in DVFS-based platforms.

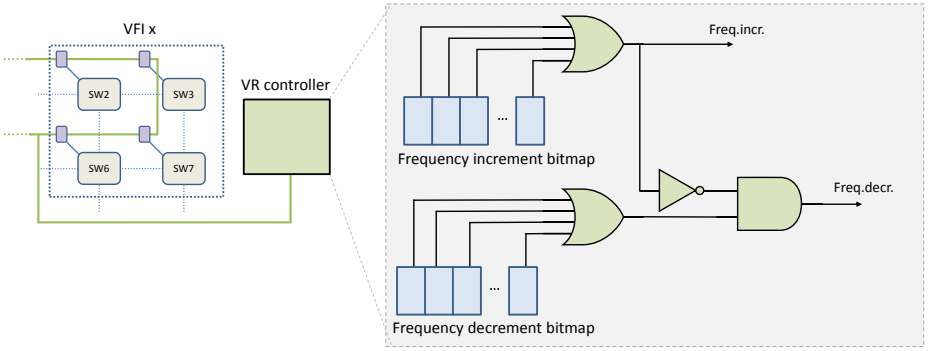


FIGURE 3.26: Voltage Regulator controller logic for ICARO-DVFS.

3.2.2 ICARO-DMSD

Following, the proposal for integrating ICARO with DMSD is described. First, DMSD is described to better understand how it works and the opportunities of merging it with ICARO and then we describe how we integrate both mechanisms to achieve a power efficient system while guaranteeing latencies for non-congested traffic.

3.2.2.1 Analysis of the DMSD DVFS Policy

The purpose of the *Delay-based Max Slow Down (DMSD)* DVFS policy is to decrease the NoC frequency and voltage as much as possible without compromising the system performance [54]. To achieve this, DMSD uses the average end-to-end latency as a performance metric, and a Proportional-Integral (PI) controller that adapts frequency and voltage so as to keep that metric close to a *latency target*. The higher the latency target, the larger is the power saving. In our experiments, we set it equal to the latency that is obtained with an injection rate 5% less than the saturation point under uniform traffic, which is obtained by running simulations in which the injection rate is increased until saturation. However, in a real system the latency target can be obtained by means of profiling.

In Fig. 3.27 an overview of an NoC provided with DMSD is depicted. Each node stores in a register the average end-to-end latency, updated each time a flit is received. Periodically, all nodes send the average latency to a given node, which computes the overall

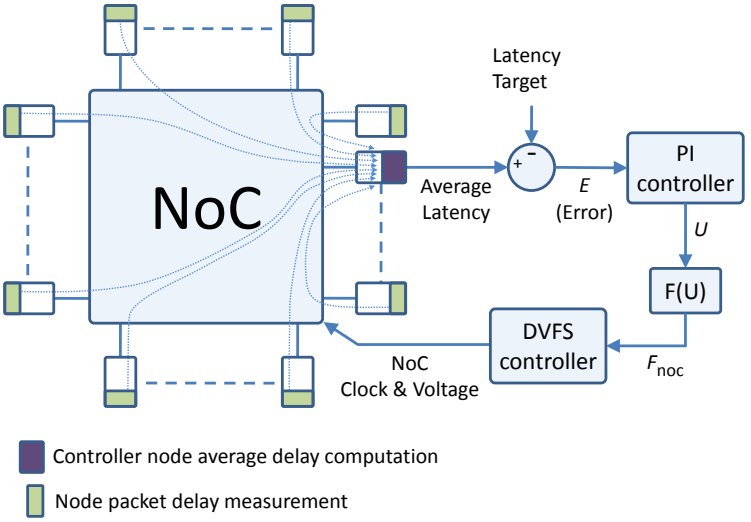


FIGURE 3.27: All nodes in the network send latency measures to the PI controller to set the new frequency.

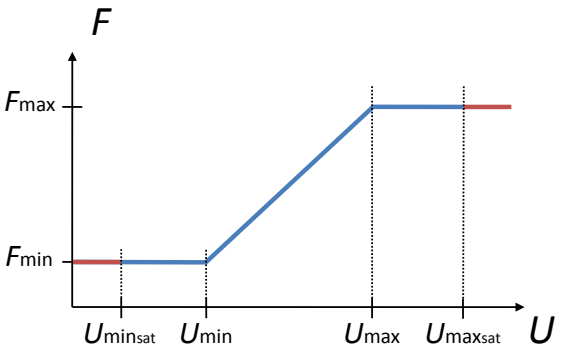


FIGURE 3.28: Conversion from U to frequency.

end-to-end latency for the whole system. In addition, this node contains the PI controller and the voltage and frequency controllers. Upon receiving all latencies, the overall latency at time n, L_n, is computed and the noise filter described by (3.1) is applied to obtain L'_n. Then, the *error* E_n is computed by subtracting the *latency target* L_t from L'_n as shown in (3.2). The error is then passed to the PI controller, which generates the signal U_n according to (3.3).

$$L'_n = \alpha\, L'_{n-1} + (1-\alpha)L_n \tag{3.1}$$

$$E_n = L'_n - L_t \tag{3.2}$$

$$U_n = U_{n-1} + K_I E_n + K_P(E_n - E_{n-1}) \tag{3.3}$$

In (3.3), K_I and K_P are the integral and proportional gains determined empirically and used to adjust the PI controller behavior while guaranteeing stability.

Finally, U is used to determine the frequency. For this, U is bounded within $U_{min_{sat}}$ and $U_{max_{sat}}$ and the range from U_{min} to U_{max} is linearly translated to frequency, as shown in Fig. 3.28. A voltage-to-frequency mapping is then used to apply the correct voltage for a given clock frequency.

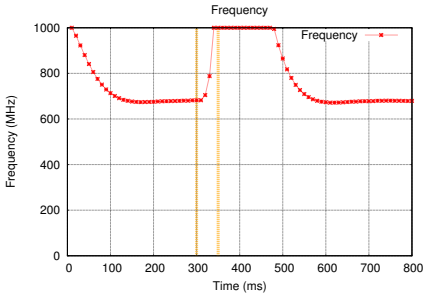


FIGURE 3.29: Frequency for DMSD under hotspot traffic.

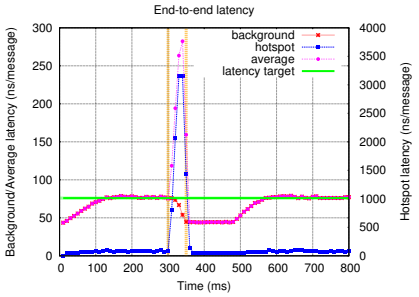


FIGURE 3.30: End-to-end latency per traffic type for DMSD under hotspot traffic.

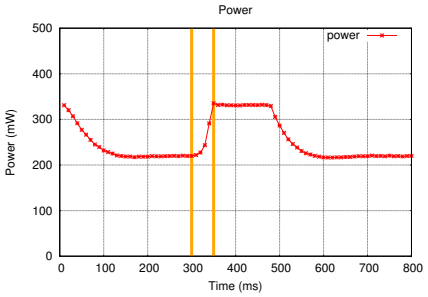


FIGURE 3.31: Power consumption for DMSD under hotspot traffic.

DMSD performs well under stationary traffic patterns [54]. As shown in Figs. 3.29-3.31, however, under hotstpot-based traffic patterns, the high intensity of a few data flows (hotspots) which are not representative of the whole system load, disrupts the DVFS strategy, leading to a waste of power by increasing the frequency and voltage unnecessarily as can be seen in Fig. 3.31.

Notice that this effect could be avoided by implementing Voltage and Frequency Islands (VFIs) [55][56]. However, this would imply extra silicon area and power to implement the VFIs separate DVFS controllers, and it would require to either know at design-time where hotspots will be located, or the ability to confine the hotspot traffic in a separate voltage island at run-time. In contrast, our approach consists in implementing a congestion control mechanism (ICARO) to detect hotspots and filter them out, regardless their location and intensities.

3.2.2.2 Implementing Congestion Management

Hotspot traffic patterns may mask the overall system performance, increasing significantly the overall latency while most of the network resources not affected by hotspots

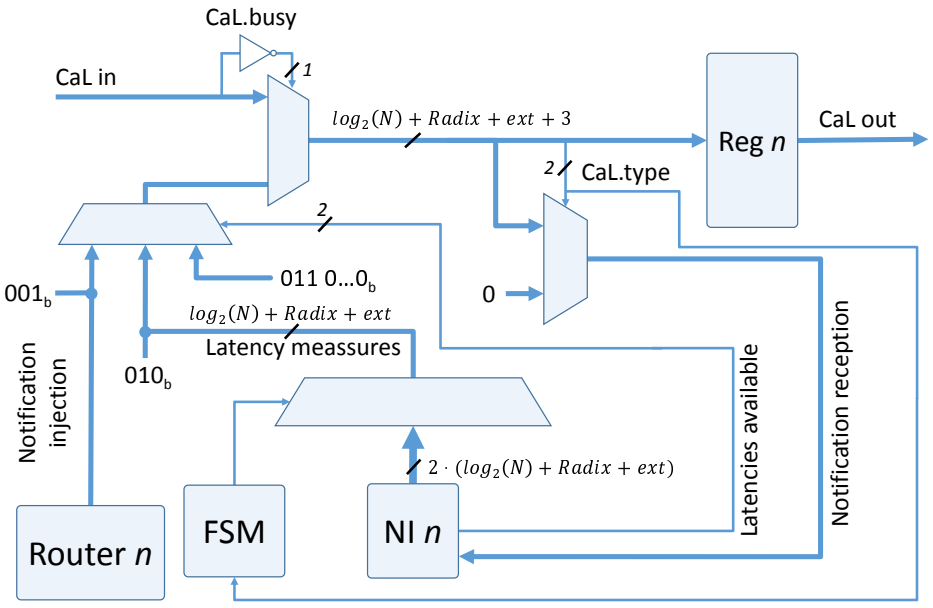


FIGURE 3.32: CaL network register associated logic for regular nodes adapted to DMSD.

may be underutilized leading DMSD to overreact under these traffic patterns as shown in Figs. 3.29-3.31. Our approach consists in combining ICARO in order to identify hotspot flows, and separating them from the rest of the network traffic (background traffic). Then, we modify the way in which DMSD collects latency metrics so that, rather than collecting latencies from any received message, our proposal collects metrics only from regular-VNs. Since ICARO effectively isolates hotspot traffic into the extra-VN, we achieve the frequency to be adjusted only for non-hotspot traffic so that DMSD achieves the non-hotspot traffic to accomplish with the latency target independently of the presence of congested traffic in the network. Following, we describe modifications performed in order to combine both mechanisms.

Delivering Latency Measurements with the CNN Network

In the original DMSD formulation, packets containing the measured end-to-end latency are sent to the controller node via piggybacking [54]. Intense congestion situations, however, may delay the delivery of those packets and the reaction of the PI controller, potentially causing the PI controller to oscillate. On the other hand, ICARO implements the CNN dedicated network, which we modify to support the delivery of those latency values in addition to congestion notifications. By doing this, the metrics necessary to set the frequency properly are guaranteed to timely arrive at destination (with low aggregated overheads, as we show in Sec. 3.2.3) through the CNN (hereafter CaL network[5]; Congestion notifications and Latencies network).

[5]The ICARO dedicated network is called CNN. However, to prevent from misunderstandings and to keep it coherent with its new additional usage, in this proposal the name for the dedicated network is changed to *CaL*

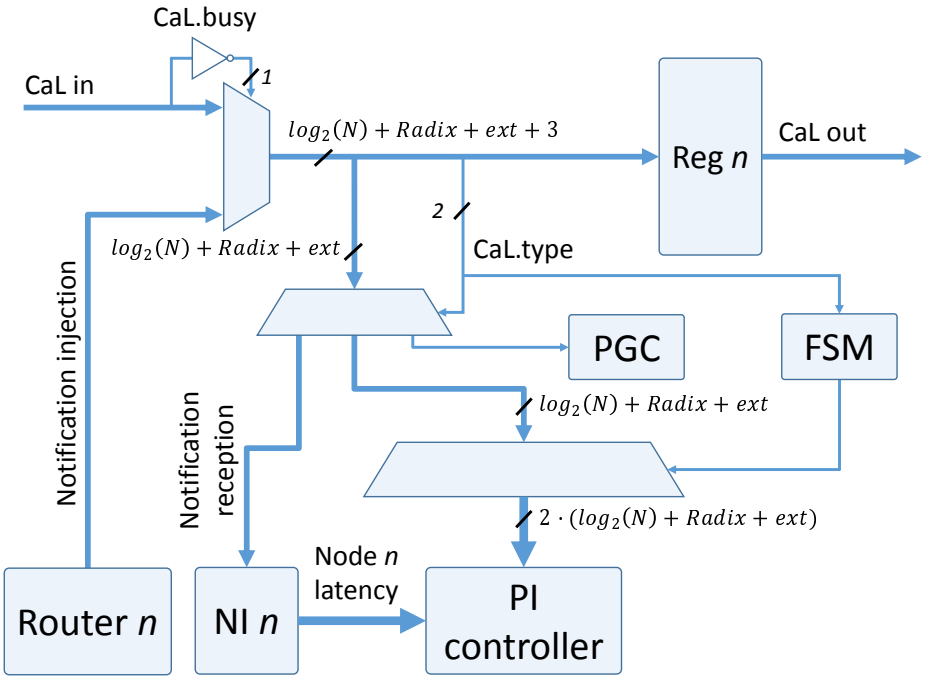


FIGURE 3.33: CaL network register associated logic for the node provided with the PI/DVFS controller adapted to DMSD.

In a DMSD-based system there are two types of nodes: those that send their latency metrics, and one that receives them. Thus, we implement two slightly different logic blocks to connect to the CaL network. Fig. 3.32 shows the logic of a typical router/NI that sends and receives ICARO notifications and sends DMSD latencies. Note that, despite ICARO notifications are sent in one cycle, we modify the logic to serialize the transmission of 32-bit latencies through the CaL network by extending its links width by *ext=8-Radix* bits. Congestion notifications can be seamlessly interleaved with DMSD latency notifications because one bit identifies the type of CaL message. DMSD notifications from different nodes are guaranteed to arrive in order since nodes will send them after a fixed (and different for each node) time offset. Similarly to the sender node in Fig. 3.32, Fig. 3.33 shows the logic for the receiver node. This node sends and receives ICARO notifications like any other CaL node, adds its own latency measurements to the received ones, and forwards them to the PI controller.

Power-Gating Extra-VN Buffers

To avoid wasting power when there is no congestion, we implement a mechanism to power-off the extra-VN buffers via a centralized Power-Gating Controller (PGC), which resides in the same node that implements DMSD. All the buffers of an extra-VN (in all NIs as well as in all routers) are powered on/off simultaneously. Power-on is easy: the PGC node snoops the CaL network and when it catches the first congestion notification it broadcasts a power-on message through the CaL network. On the contrary, power-off is not trivial. Snooping the CaL network in search of the "end of congestion" messages

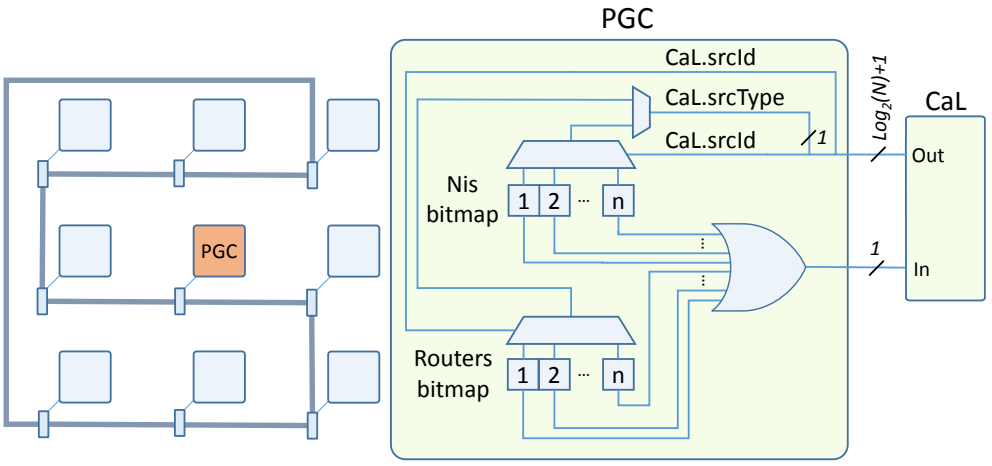


FIGURE 3.34: Power-gating controller.

is not a valid strategy because there might still be messages in the extra-VN, either in the NIs pending to be injected, in the routers on their way to destination, or both. Therefore, to safely turn off the extra VNs, the PGC must be informed through the CaL network by both all NIs and routers about their detection of a *congestion-free* situation:

To make sure that no congested traffic will be injected into the network from a given NI, two conditions must be satisfied. First, the extra-VN buffers must be empty. In addition, the NI congestion notification board must be clean (no new notifications). If both conditions are met, the NI notifies its *congestion-free* status to the PGC with a special message sent through the CaL network.

At routers the mechanism is simpler. Each time the *extra-VN* buffer utilization increases from 0 to 1, the router sends a message to the PGC to inform that is storing congested traffic. On the other hand, when the buffer utilization decreases from 1 to 0, the router sends another message to inform that is *congestion-free*.

The PGC is provided with two N-width bitmaps, where N is the number of nodes in the network: one bitmap for the NIs and one for the routers. These bitmaps are updated each time a NI or a router notifies the PGC about its status (0=no congested traffic stored, 1=congested traffic stored). In this way, the PGC has a complete *congestion picture* of the network. When the PGC detects that all NIs and routers are *congestion-free*, it commands to power-off the extra-VN buffers; otherwise, it commands to power-on the buffers. Fig. 3.34 sketches the PGC bitmaps, the logic to power-on/off the buffers, and its connection to the CaL network. We quantify the advantage of using power-gating in Sec. 3.2.4.2.

Note that area and power consumed by the PGC are negligible since its implementation only requires N-width demultiplexers, 1 N-width multiplexer, 1 N-width NOR gate and $2N$ registers for a complete mesh.

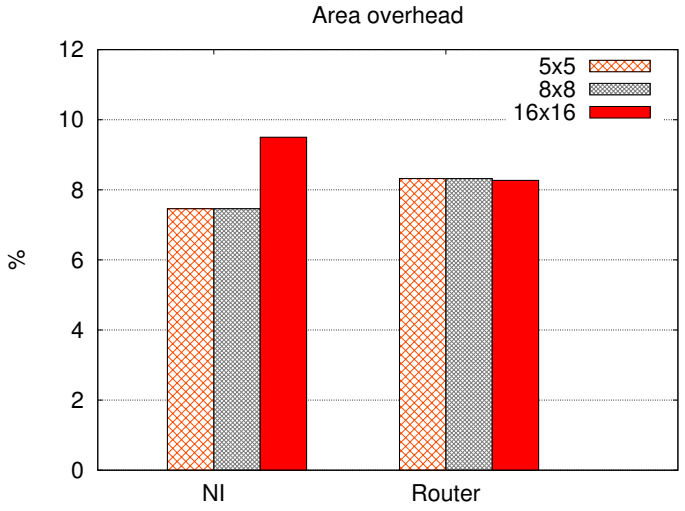


FIGURE 3.35: ICARO-DMSD area overhead of different meshes.

3.2.3 Area Overhead Analysis

The bars in Fig. 3.35 illustrate the area overhead for a NI and a router with support for ICARO, with respect to a baseline implementation (no DMSD, no ICARO)[6]. The results have been obtained after synthesis on our 28-nm technology, in the conditions of Tab. 3.7, except for the mesh size that we let vary. We notice that the overhead is small, less than 10%, even for the case of a large 16×16 mesh.

3.2.4 Evaluations

3.2.4.1 ICARO-DVFS

As in previous evaluations, our simulator models a 4-stage pipelined router: IB, RT, SA, X for all routers but those defining the boundaries of each VFI. In the case of routers at the VFI boundary, a *coupling IB stage* (see Section 3.2.1.2) is implemented with two substages: *IB_W* (writing data) and *IB_R* (read data). *IB_W* stage writes incoming flits from the upstream router, therefore, this substage works at the frequency of the VFI to which the upstream router belongs. However, as the *IB_R* stage reads from the router buffers, this stage runs at the frequency of the router implementing it. In Table 3.6 further configuration details are shown.

Regarding traffic patterns, as ICARO is a proposal intended to deal with irregular, bursty, hotspot-prone traffic patterns, we drew up a combined traffic pattern. This

[6]We obtained overhead results only for ICARO since DMSD and the PG controller overheads are negligible compared to the ICARO's overhead.

Topology	8x8 2D regular mesh
Routing policy	XY
Switching technique	Wormhole (flit-level)
Flow control	credits
Flit size	128 bits
Message size	5 flits
Switch queue size	16 flits
VFIs	4 VFIs (4 nodes each)

TABLE 3.6: Common simulation configuration.

traffic pattern is composed of a light uniform background traffic (at a data rate of 0.01 flits/ns) and a hotspot component. Hotspots consist in several nodes receiving each one high data rates from 4 nodes (4-to-1 hotspots). Hotspots are active only from time 20ms to 40ms. In this way, we have a background traffic which generates no congestion, plus an aggressive traffic which causes congestion, causing HoL-blocking to the background traffic. This compound traffic pattern emulates environments where light data flows (i.e.: cores running applications with light data demand) share the NoC resources with heavy traffic generated by nodes running high data demand applications or hardware accelerators which tend to generate heavy data bursts, causing congestion as well.

Simulations with DVFS are performed using real voltage, frequency and delay values shown in Table 3.5. We implement 4 VFIs each one containing 16 nodes. For power consumption measures we use Orion 3.0[57].

First, two configurations without DVFS are considered: *minFreq* for the chip running at the minimum frequency (1.54GHz) and *maxFreq* for the chip running at the maximum frequency (3.074GHz). Then, results for a DVFS scenario (without ICARO) are shown. Finally, results for the three ICARO-DVFS versions are shown. It is worth recalling that the main ICARO goal is to keep background traffic unaffected when dealing with congestion-prone traffic, such as the simulated hotspot traffic. For that reason, results for background and hotspot traffic are shown separately.

Figure 3.41 shows average network latency results. For the DVFS cases some peaks in latency can be seen when congestion background starts and ends. These peaks clearly show the overhead derived from the VR taking some time to perform the DVFS level changes. It is worth mentioning that the first peak is higher as the VR takes more time to change from the lowest frequency to the highest. After some analysis, we decided to increase from the minimum frequency to the maximum one since this involves only one transition, incurring much less penalty with respect to a step-by-step increase. As expected, the scenario with DVFS improves in power consumption the no-DVFS scenario, but at the cost of increasing network latency (Figure 3.43) and decreasing throughput

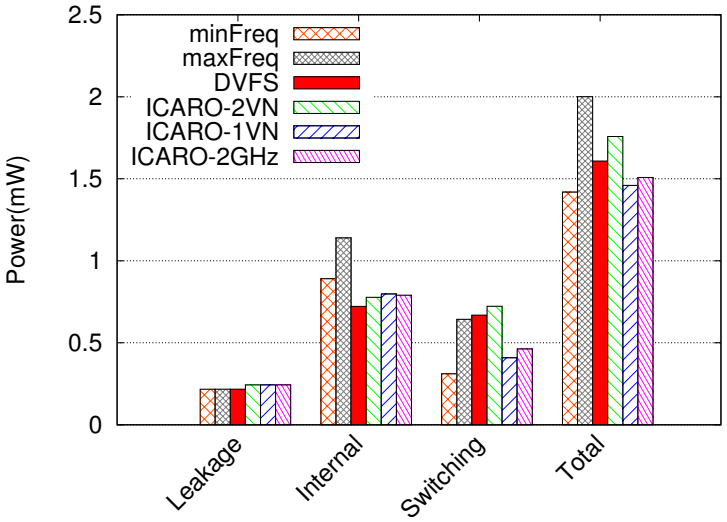


FIGURE 3.36: Final power consumption.

(Figure 3.44). However, according to Figure 3.36, power consumption saving is more significant than performance degradation.

As can be seen in Figures 3.41, ICARO proposals separate effectively background traffic from the hotspot one, preventing the HoL-blocking effect over the former. As expected, ICARO-2VN achieves the highest improvements in network latency (up to 43%) since it takes into account all VNs to trigger the frequency-increment mechanism, and it is able to increment frequency to the maximum, but at the cost of increasing power consumption (∼8%). Nevertheless, despite being the ICARO proposal with the highest power consumption, this case still keeps power consumption below "DVFS-alone" levels while improving latency.

Regarding the results for ICARO-1VN, note that only *regular-VNs* (VN 0) is taken into account to increase the VFI frequency and only background traffic is forwarded through this VN, so that this slight traffic flow is not enough to trigger the frequency-increment mechanism and all VFIs end up working at the minimum frequency. Despite running at the minimum frequency, we can see that the background network latency is kept in similar values to the DVFS case working at highest frequency. This is achieved by separating both traffic types, so preventing the HoL-blocking that the hotspot traffic could cause. In addition, as frequency is the lowest allowed, power saving is maximum, achieving an improvement of 26%. However, as the system is working at the minimum frequency, throughput is lower than other cases that are running at higher frequencies. Nevertheless, background traffic achieves acceptable latency values.

Finally, ICARO-2GHz case could be considered as the best trade-off proposal. It takes into account all the available VNs but it is only allowed to increment frequency to the next step from the lowest frequency. This proposal does not achieve the best results in

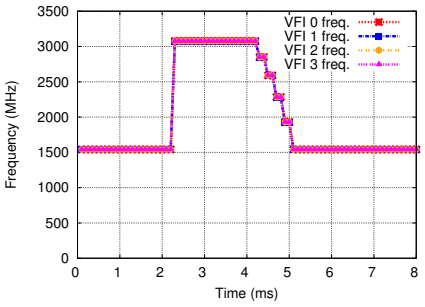


FIGURE 3.37: VFIs frequencies for DVFS without ICARO.

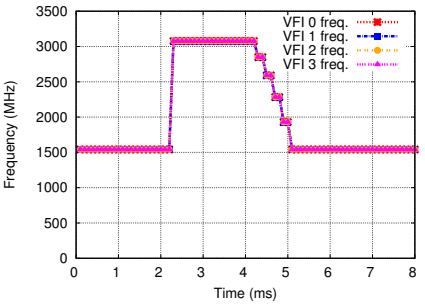


FIGURE 3.38: VFIs frequencies for ICARO-2VN.

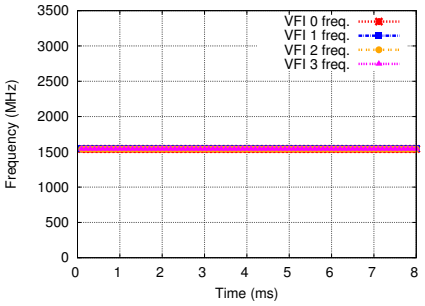


FIGURE 3.39: VFIs frequencies for ICARO-1VN.

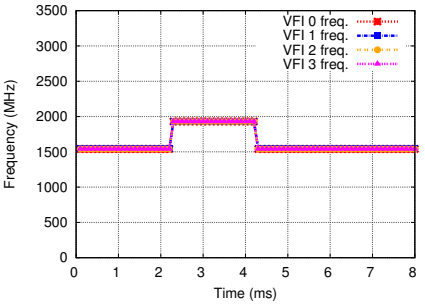


FIGURE 3.40: VFIs frequencies for ICARO-2GHz.

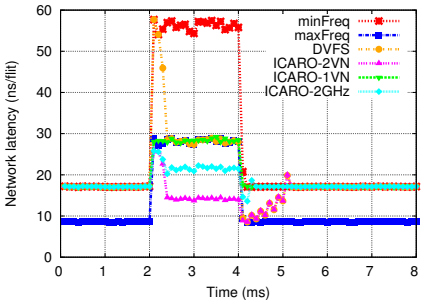


FIGURE 3.41: Network latency for background traffic.

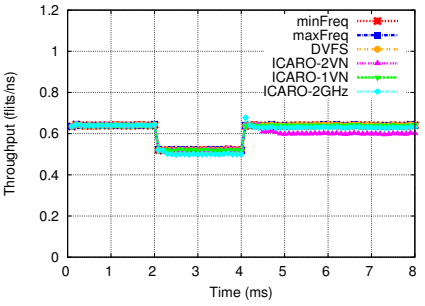


FIGURE 3.42: Throughput for background traffic.

network latency and throughput, but nevertheless it improves the "DVFS-alone" case by 19% with a significant power-saving improvement (20%) with respect to DVFS, due to the lower frequency used.

3.2.4.2 ICARO-DMSD

In this section we first report simulation results obtained in the baseline configuration of Tab. 3.7. Then, we report results obtained with a sensitivity analysis in which we varied several configuration parameters to check the robustness of our solution. Note that, for our experiments DMSD as well as ICARO-DMSD are provided with the same amount

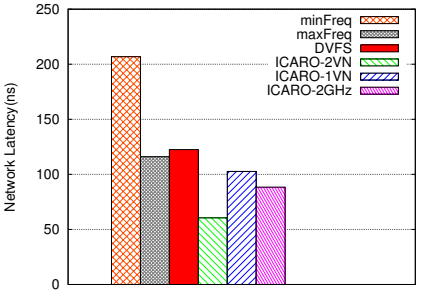


FIGURE 3.43: Final net. latency (all traffic).

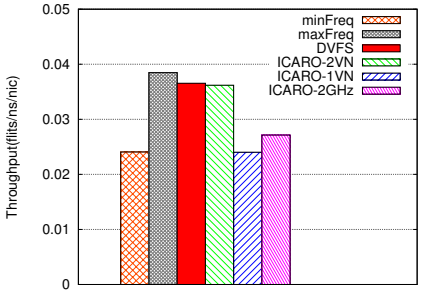


FIGURE 3.44: Final throughput (all traffic).

of VNs in order to compare both solutions with the same amount of resources, providing each VN with the same amount of VCs. However, since DMSD does not require several VNs to work properly and these additional resources may affect negatively to its power consumption we perform an additional experiment comparing against the baseline with only 1 VN. Note that, since the goal of our approach is to keep the background latency around the latency target, for better understanding, hotspot latencies have been omitted in the graphs. Also note that the hotspot start/stop time is highlighted with vertical bars.

As in our previous evaluations, for calculating power consumption we use Orion [57]. However, Orion does not directly support the industrial 28-nm CMOS technology that we used for the implementation of routers with support for congestion management and buffer power-gating. By using the post-synthesis results of our RTL version of the router, we modified Orion in such a way that its results are compatible with our technology. Moreover, we added the support for including the effect of buffer power-gating in the computation of power consumption.

Figs. 3.45-3.47 compare the *DMSD* and the *ICARO-DMSD* cases in terms of latency, frequency, and power, in the baseline scenario. Notice that, to properly compare the two cases, the two systems have the same total buffering resources. Note also that, in the case of ICARO-DMSD, the *extra-VN* is composed of as many VCs as the *regular-VNs*.

Since ICARO effectively separates the background traffic from the hotspot one, DMSD can effectively measure only the latency of the background traffic. Therefore, thanks to the PI controller, DMSD keeps the latency of the background traffic around the 76-ns *latency target*, as shown in Fig. 3.45. In fact, as Fig. 3.46 shows, the NoC clock frequency is not influenced anymore by the activation of the hotspot traffic. This, in addition to the use of power-gating, results in a significant improvement of the power consumption, as shown in Fig. 3.47. When the hotspot is not active (from time 0 μs to 300 μs, and then again after around 380 μs), the extra-VN buffers are powered-off, resulting in lower power for the *ICARO-DMSD* case. When the hotspot is active, the extra-VN buffers are

TABLE 3.7: Simulations configuration.

Network configuration	
Topology	8x8 2D regular mesh
Routing policy	XY
Switching technique	Wormhole (flit-level)
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4 per Virtual Network
DMSD configuration	
Frequency range	333 - 1000 MHz
Voltage range	[0.56, 0.9] V
K_i, K_p	0.025, 0.0125
U saturation range	[-15, 15]
α	0.7

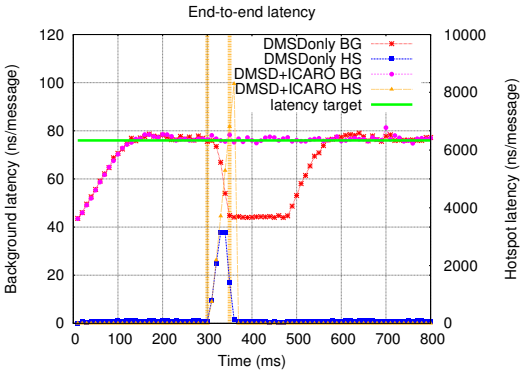


FIGURE 3.45: End-to-end latencies for the background and the hotspot traffic.

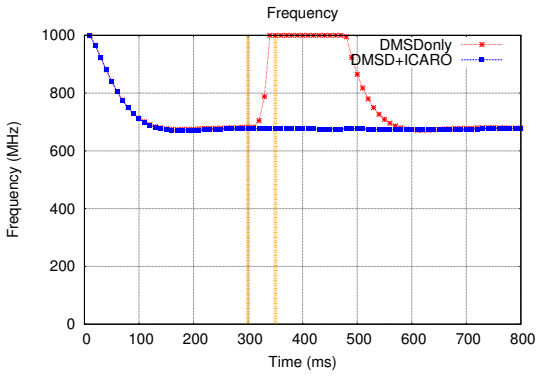


FIGURE 3.46: Frequencies for DMSD and ICARO-DMSD.

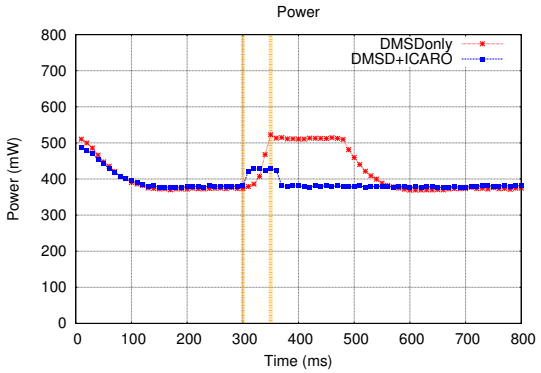


FIGURE 3.47: Power consumption for DMSD and ICARO-DMSD.

switched on, hence the power increases. Still, since the clock frequency in the *ICARO-DMSD* case is less than in the *DMSD* case, the power consumption is also significantly

TABLE 3.8: Robustness analysis scenarios configuration.

Scenarios							
Label	Mesh Size *(nodes)*	Queue Size *(flits)*	VCs	Msg. Length *(flits)*	Num. HS	HS Dur. *(ns)*	Lat. target *(ns)*
Baseline	8x8	4	4	10	1	50us	76
5x5	5x5	4	4	10	1	50us	66
16x16	16x16	4	4	10	1	50us	105
qs2	8x8	2	4	10	1	50us	79
qs8	8x8	8	4	10	1	50us	81
qs16	8x8	16	4	10	1	50us	72
vcs2	8x8	4	2	10	1	50us	60
vcs8	8x8	4	8	10	1	50us	97
ml5	8x8	4	4	5	1	50us	62
ml20	8x8	4	4	20	1	50us	96
2HS	8x8	4	4	10	2	50us	76
3HS	8x8	4	4	10	3	50us	76
short	8x8	4	4	10	1	25us	76
large	8x8	4	4	10	1	100us	76

reduced.

To validate our results under different network configurations, we changed several network parameters: mesh size, router buffers queues size, number of virtual channels, message length, number of hotspots, and hotspot duration. All cases analyzed are described in Tab. 3.8, in which each case is identified with a label that is used next in the graph keys. As Fig. 3.48 shows for all the configurations analyzed, in the *ICARO-DMSD* case the background traffic correctly tracks the prescribed target, hence avoiding the excessive power consumption that characterizes the reference *DMSD* case. Note that in the *hotspot duration* graph the three different hotspot ending times are highlighted with different colors. Please note that the *latency target* value for a given scenario depends not only on the saturation point, which is highly correlated with the system configuration, but also on the latency curve gradient. Therefore, in some system configurations the calculated *latency target* seems not to follow an intuitive progression like in the VCs analysis graph shown in Fig. 3.48.

Fig. 3.49 summarizes the improvement of power consumption of the *ICARO-DMSD* case, in all the configurations of Tab. 3.8. Two different improvement values are reported. The first one is due to the *extra-VN* power-gating (*no-HS* in the graph), measured at time $290\,\mu s$ (just before the hotspot activation); the second one corresponds to the power-saving during the hotspot duration (*HS* in the graph) and is calculated by averaging the power spent from time $300\,\mu s$ to time $600\,\mu s$, since this is the time range in which the hotspots affect any of the cases analyzed. Note that the power overhead due

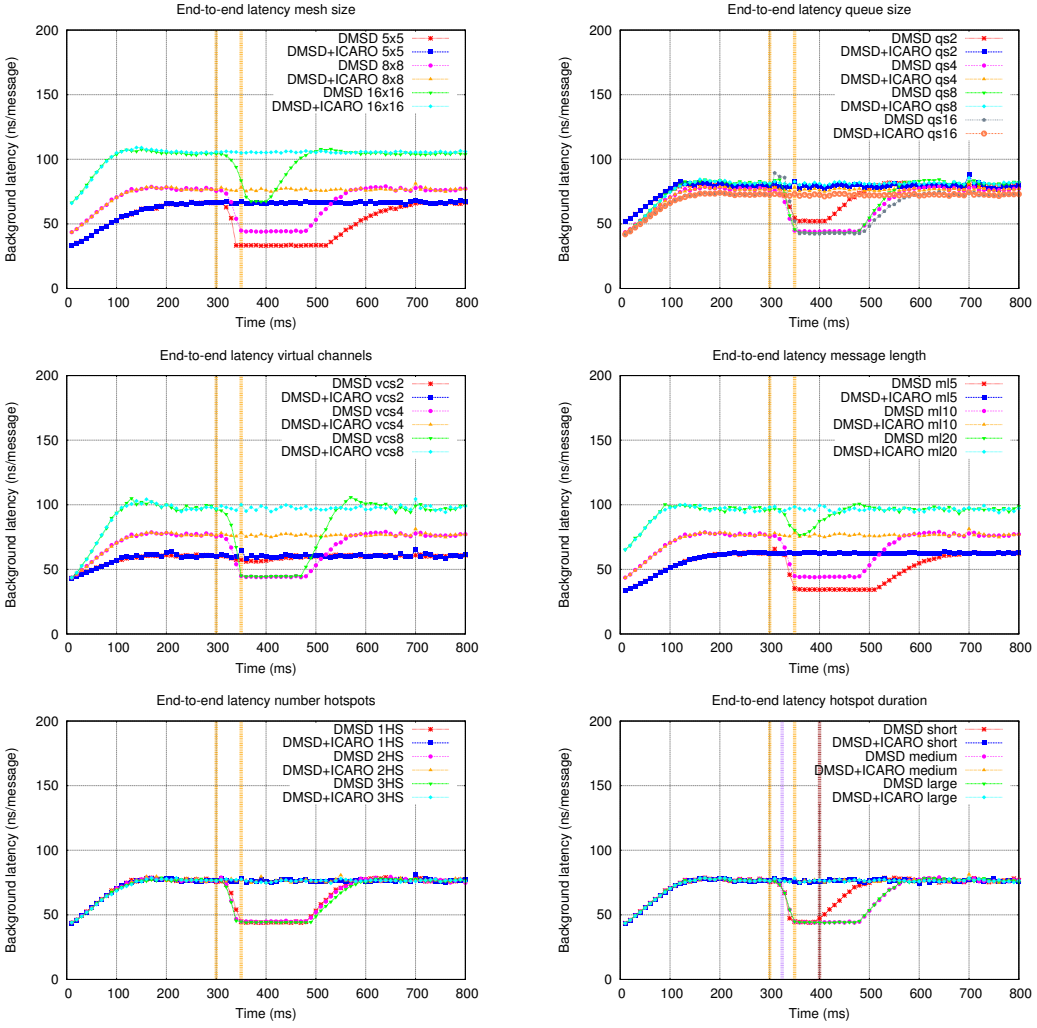


FIGURE 3.48: End-to-end latency for different configuration parameters

to the additional hardware required by our proposal is already included in the power consumption graphs.

As Fig. 3.49 shows, for all the cases considered, the combination of DMSD and ICARO leads to a significant power improvement over the DMSD baseline when hotspot is active. When no hotspot is active, by switching the *extra-VN* off we achieve up to 38% power saving and an average of 28%. When congested traffic is detected, ICARO manages this sort of traffic and DMSD tunes the frequency properly saving up to 53% power consumption and 38% on average. In the results obtained when the hotspot is present, we observe a larger variance. This is expected as, for calculating the average, we take values from the same range of time for all cases but the duration of the effects of the hotspots are not the same for those cases, therefore, the weight of those values over the average is not the same.

In the final experiments we analyze our proposal against DMSD provided with 1VN. Unlike ICARO-DMSD, DMSD does not require several VNs to perform properly, so we

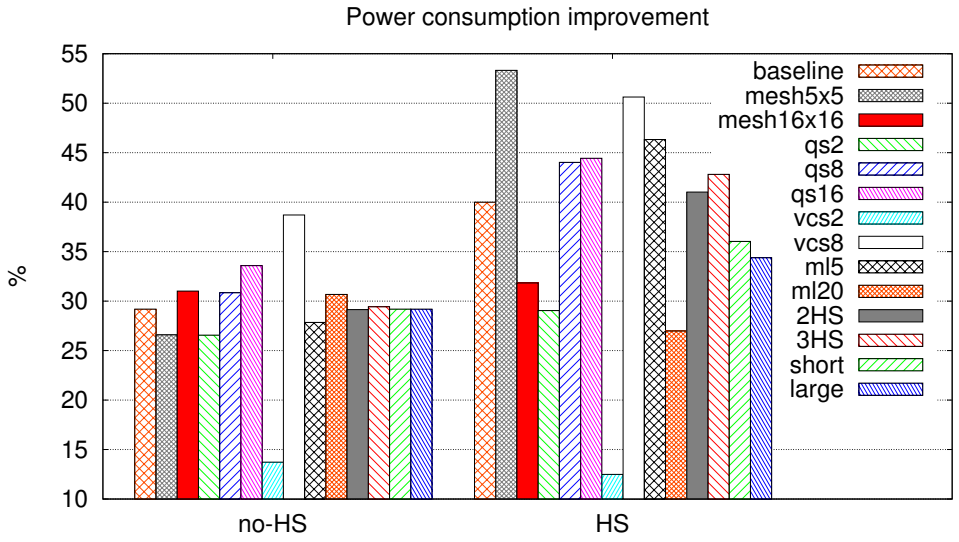


FIGURE 3.49: Power consumption improvement with respect to DMSD for all configurations.

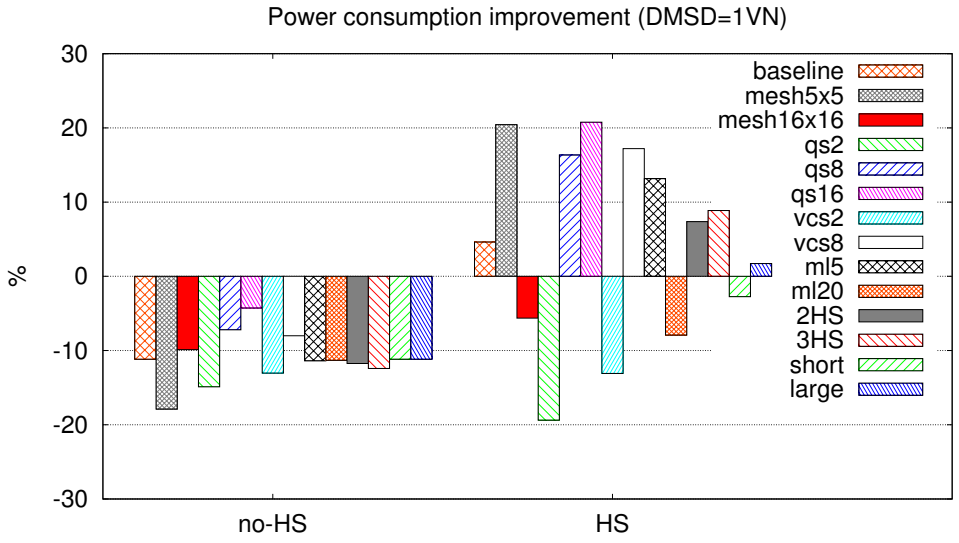


FIGURE 3.50: Power consumption improvement with respect to DMSD (provided with 1VN) for all configurations.

performed the same robustness analysis shown above but providing DMSD with 1VN. Nonetheless, as ICARO-DMSD does not require the *extra-VN* to be provided with several VCs, for this simulations we configured ICARO-DMSD with the same number of VCs for the *regular-VN* as the DMSD case and only 1VC for the *extra-VN*. The power results in Fig. 3.50 show that in absence of congestion, ICARO-DMSD consumes more power than DMSD due to the ICARO logic power consumption. When hotspot is active, however, despite the additional buffers ICARO-DMSD saves a significant amount

of power under the most part of the analysis, achieving up to 20% power saving. Nevertheless, some scenarios present characteristics (amount of resources, message length, etc.) for which DMSD does not overreact to keep the latency under the target, resulting in less power consumption for DMSD. Still, the congestion in those scenarios triggers the ICARO mechanism, causing to switch the extra-VN buffers on, increasing power consumption, and ultimately reducing the power saving compared to the baseline.

3.3 Reducing Buffers Leakage Power

Previously we proposed congestion management mechanisms and combined them with power-saving techniques based on decreasing the clock frequency. However, as technology scale goes further, leakage power is becoming an important contributor of the overall power consumption. As described in previous sections, ICARO requires to implement special queues intended to confine congested traffic. Since these buffers are only used when congestion is detected in the network, seems clear that to keep them switched on represents a waste of power. In previous works we proposed other approaches to deal with this by power-gating *extra-VN* buffers but following a very simple strategy not properly adapted to ICARO. Because of this, we developed PAPM, a Path-Aware Power Management which we integrate in ICARO (ICARO-PAPM) and then re-design it to be a general standalone power-gating mechanism for NoCs (PAPM).

3.3.1 ICARO-PAPM

3.3.1.1 Overview

The goal of our proposal is to reduce power consumption in ICARO by implementing a new buffer power-gating mechanism (PAPM) for the *extra-VN* queues. Essentially, PAPM will power on only those queues which are necessary according to the CPs already detected. The key idea behind PAPM is that not all buffers are always needed for separate specific flows. Since deterministic routing schemes imply that a given flow from a source to a destination will be always forwarded through the same path, we can easily determine which buffers are needed for delivering such flow (those along that path). In this way, we can safely power on or off router buffers dynamically to fit the current traffic pattern requirements.

However, our proposal requires to know only those flows crossing CPs, as they are the ones using extra-queues. Each end-node, by inspecting its local memory can know the current CP locations. From that information we need to deduce the affected flows.

In addition, one key aspect of PAPM is related with the propagation of power-gating signals. Most of the current proposals for power-gating stand up for driving the modules

power signals by means of dedicated wires. This strategy suffers from scaling issues in large network sizes. Instead, since PAPM needs to power buffers on/off by building data paths according to the routing policy, it follows a strategy in which buffers are powered on in the same order a given congested message would follow through its path to destination. Taking this into account, we design PAPM to send the powering on/off signals as part of regular single flit messages sent through the regular network.

3.3.1.2 PAPM for ICARO

Each extra-VN buffer belonging either to a router port or to a NI must be powered on only if congested traffic would potentially be delivered through that port or NI. In the case of NIs, it is quite simple since they own the ICARO cache memory in where *CPs* are stored. As soon as a NI allocates a CP (the cache is not empty), the NI becomes a potential injector of congested traffic. Therefore, its extra-VN buffer must be powered on.

Regarding router buffers at input ports, to know whether an extra-queue needs to be powered on or off becomes more difficult. Figure 3.51 shows the NIs subset (NIs 6, 7 and 8) reaching the south port of router 4 (we assume XY routing). This means that each buffer at each router must be powered on when any of those NIs has congested traffic to inject and the congested point is along those paths. In the same way, each extra-VN buffer must be powered off when none of those NIs has congested traffic to inject through those points. To manage this, when a NI stores a new *CP*, it sends a special message (*allocation message*; *AM*) to the first node reachable through the *CP* port (node 1 in Figure 3.51). This message will increase by 1 a counter stored at each input buffer at each router along the path until its destination. The extra-VN buffer is powered on when the counter is greater than zero and is powered off otherwise. In the example shown in Figure 3.51, the counter at south input port at router 4 will have a value of 3 (one for each NI).

When a given NI receives a non-congested notification, the CP is removed from the cache memory and sends a message (*deallocation message*; *DM*) to the first node reachable through the CP (node 1), causing the buffer counters along the path to be decremented by 1. In this way, when all NIs able to reach a given buffer deallocated the CP, the buffer counter will reach zero, powering off the extra-VN buffer. Note that, *AM* messages are injected through regular-VN buffers (the extra-VN buffers could be powered off), while *DM* messages are sent through the extra-VN in order to guarantee that no out of order delivery arises between the *DM* and the congested data being forwarded through the extra-buffers.

Note that, although there are works proposing to switch off portions of buffers as in [58], in this case is useless since ICARO-PAPM switches off *extra-VN* buffers, which are

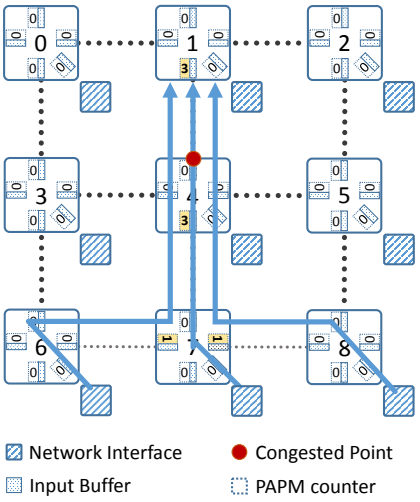


FIGURE 3.51: Network Interfaces reaching south port of router #4.

intended to deliver huge amounts of traffic (congested traffic). Therefore, it is expected to require the whole buffer when the buffer is needed.

3.3.1.3 Selective Broadcast

Once a *CP* is detected, ICARO reallocates all traffic traversing that *CP* into the extra-VN buffers, keeping the traffic in this VN until it arrives to destination, regardless the path followed after crossing the *CP*. It means that, for a given *CP* and source, there is only one path to reach the *CP* but, beyond the *CP*, congested flows might follow different routes, therefore, all extra-VN buffers along each possible path must be powered on for the given source and *CP* pair.

To address this issue we propose a *selective broadcast* mechanism. It consists in modifying the routers behavior when forwarding *AM/DM* messages. These messages are forwarded as unicast messages until they arrive to a *CP*. Once a given *AM/DM* message crosses the *CP* (in the next router), the message is treated as a regular broadcast message. *AM* and *DM* messages are single flit messages, avoiding deadlock issues in wormhole switched networks when combined with broadcast support.

This mechanism is implemented by setting the next reachable node after the *CP* as the *AM/DM* message destination. These messages are provided with two bits: the *MCactive_bit* and the *MC_bit*. The first one is set to 0 by default and is enabled only when it arrives to the destination router (the next router reachable through the *CP* port) and the *MC_bit* is set to 1. When both bits are set, the message is treated as a broadcast message. Since the broadcast mechanism works by duplicating the main message, it will carry this bit to all its forked messages and this process is replicated by each forked message along its path to its destination, crossing all reachable routers

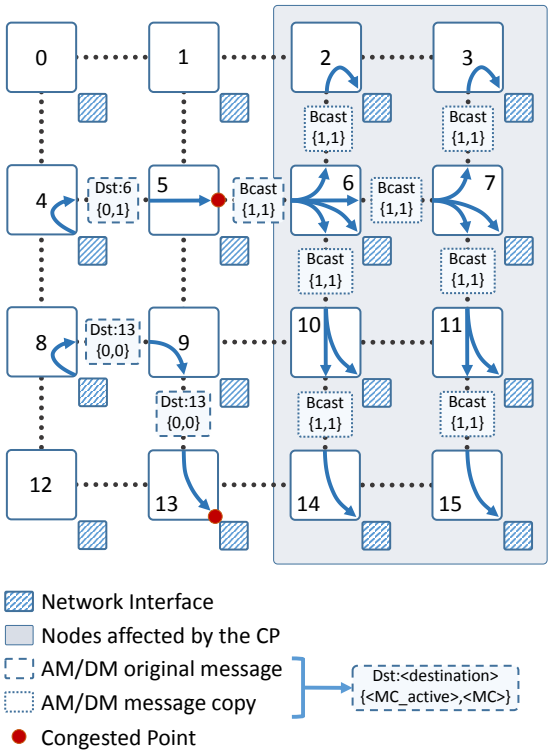


FIGURE 3.52: PAPM messages copies destinations.

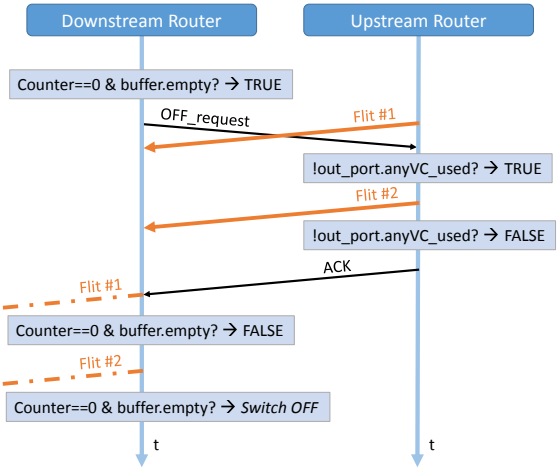


FIGURE 3.53: Buffer powering on/off protocol.

starting from the *CP*. An example of an *AM/DM* message being delivered can be seen in Figure 3.52. The *MC* bit, when reset, disables broadcast operation. This is useful when the CP is detected at an end-point. By forcing the *MC* bit to zero, the message is not broadcasted at the last router where the end-point is connected to. Two examples of the use of *MC_active* bit and *MC* bit are shown in Figure 3.52 for two different CPs.

3.3.1.4 Flow Control

When using buffer power-gating mechanisms, one of the key challenges consists in avoiding race conditions. Routers are unaware of the buffers state of their neighbors, therefore, communication between adjacent routers becomes essential to know when is safe to forward data to the next buffer. For this purpose, PAPM implements a simple handshake protocol between adjacent routers. When a given buffer has to be powered off (its counter reached 0 and the buffer is empty). The router sends an *OFF_request* to the upstream router for the corresponding port. Next, when the upstream port receives the *OFF_request* sets a bit that disables the output port for being selected for delivering more flits. In addition to this, the router checks for flits having already assigned any VC for such output port. If not, an *ACK* is sent to the downstream router. If there is any flit owning a VC for such output port, the VA/SA stage stops assigning VCs for that output port and the ACK is sent when no more messages own a VC for that output port. Finally, when the *ACK* is received by the downstream router, the buffer is powered off. However, it is worth to note that, between the *OFF_request* and the *ACK* arrival, some flits might arrive from the upstream router, and they must be forwarded. Therefore, actually, after receiving the *ACK*, the downstream router checks the corresponding buffer and powers the buffer off only if the buffer is empty. Otherwise, the buffer is marked as *requested to be powered off (RSO)* and, finally, the buffer is powered off when it gets empty. However, due to congestion transients, a buffer being powered off due to a recently disappeared CP, may again be requested to be powered on because the allocation of a new CP reachable also through the same path. To support this, the powering off algorithm will cancel the RSO state in case of the counter to be increased from 0 to 1 due to reception of an AM message while in RSO state. An example of this protocol is depicted in Figure 3.53.

The protocol for powering a buffer on requires only to send an *ON_request* handshake message from the downstream router. Since to power a buffer on can not generate any race condition, this message only causes the upstream buffer to enable that port to be selected in the VA/SA stage, allowing data to be forwarded to the downstream router through the extra-VN. No response from the upstream router is needed.

3.3.2 PAPM

In the previous section we described PAPM as a sub-mechanism of ICARO in order to alleviate the leakage power consumption of the additional resources required by ICARO. Due to the potential of PAPM, we adapted it to work in a standalone way, as a general-purpose power-gating mechanism, independently of ICARO. Therefore, following we describe the standalone proposal: PAPM.

The PAPM method works at the granularity of paths. One path is defined by the source and destination end-nodes connected through the NoC. Those nodes use a fixed path (we assume deterministic routing) to communicate. Along this path, a set of buffers are used to flow control the advance of the traffic. Therefore, a path can be seen as a chain of buffers.

PAPM manages the status of all buffers along a path. Buffers can be powered on or off. Whenever a path needs to be used, the source end node injects a control message, referred to as ABP (Activate Buffers Path) similar to the *AM* message sent in ICARO-PAPM, in order to power on all the buffers along the path. Those buffers will then be kept on during the transmission of traffic along the path. When the source node has no more traffic to inject or when it decides to temporarily switch off the path (to save energy), then the node injects a similar message, referred to as DBP (Deactivate Buffers Path), which is the analogous to the *DM* message in ICARO-PAPM, in order to power down buffers along the path.

One important aspect of PAPM is to be fast enough when powering on buffers. Indeed, those buffers need to be on for the transmission of incoming messages. To speed up this process, the PAPM method will rely on a lightweight fast network implemented as a bidirectional ring. This network, referred to as *Activation Network (AN)*, allows powering on all buffers along a path.

The power down process in PAPM (DBP message) works, however, in synchrony with the transmission of messages. Indeed, PAPM will inject the DBP message through the regular NoC network, potentially switching off buffers along its way to final destination.

One key aspect of PAPM is the management of shared buffers by concurrent flows. Indeed, two non-disjoint paths in the network will share some input ports, and thus, buffers. Activating and deactivating buffers for one path does not have to conflict with the expected buffer status of the other path. Indeed, buffers need to be powered on if any of the paths sharing the buffer are active. Figure 3.54 shows the case.

To address the sharing buffers issue, PAPM will rely on an internal counter strategy on every router input port. The counter will increase by one for every ABP message received addressing that buffer. Accordingly, the counter will decrease by two (explained in Section 3.3.2.2) for every DBP message received through the associated input port. The input port will be activated (powered on) based on the counter value.

When a node allocates a message for a given destination, the NI PAPM module checks the destination in order to know whether this path is available (switched on) or not.[7] This information is stored in the *Active Paths Bitmap* by means of a bit for each destination node (1=path active, 0=path inactive). If the path is currently active, no additional

[7]Note that there is no way to certainly know whether the whole path is effectively on/off since other nodes may share parts of the path. A given node is only able to certainly know whether it has requested to switch the path on or off.

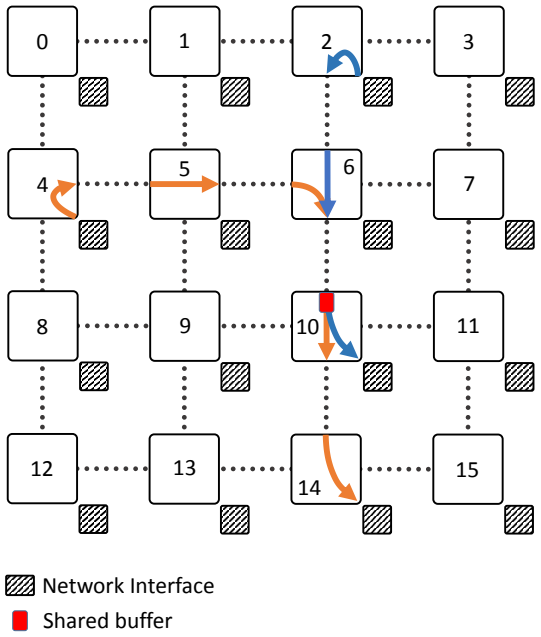


FIGURE 3.54: Flows sharing buffers along their paths.

action is required to send the message. Otherwise, an ABP is sent through the AN network in order to switch on the path, being marked as *active* in the paths bitmap. Note that the path may actually be switched on because another node has requested the same path to be switched on. Once the message is sent, PAPM checks whether the path must be switched off again. If so, PAPM sends a DBP message through the regular network in order to switch the path off and the path is marked as *inactive*.

In addition to switch buffers off, to save more power, PAPM monitors at each router the state of all buffers. When all buffers are off, since the rest of the router logic is no longer needed, the whole router is switched off as well. Accordingly, when any of the buffers are switched on, the router logic is also switched on.

3.3.2.1 Router Implementation

Figure 3.55 shows the implementation of PAPM strategy on the baseline router assumed. The router implements the standard logic blocks, namely input buffer, routing unit, virtual channel allocator, switch allocator and crossbar. The PAPM strategy impacts mainly on the input buffer strategy. A counter and a logic block is added to update the counter based on the arrival of ABP and DBP messages. Notice that DBP messages arrive through the input port associated to the buffer whereas ABP messages arrive through a lightweight new input port for the router. The added control network will deliver ABP messages through that port.

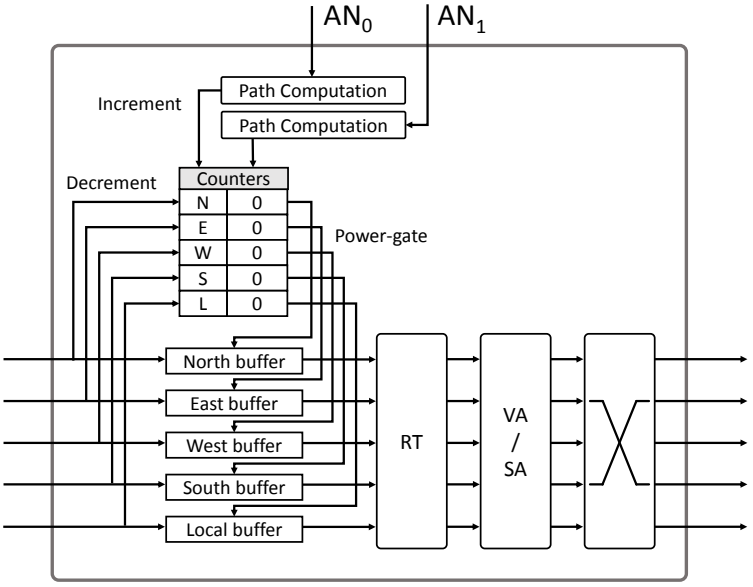


FIGURE 3.55: Router implementation.

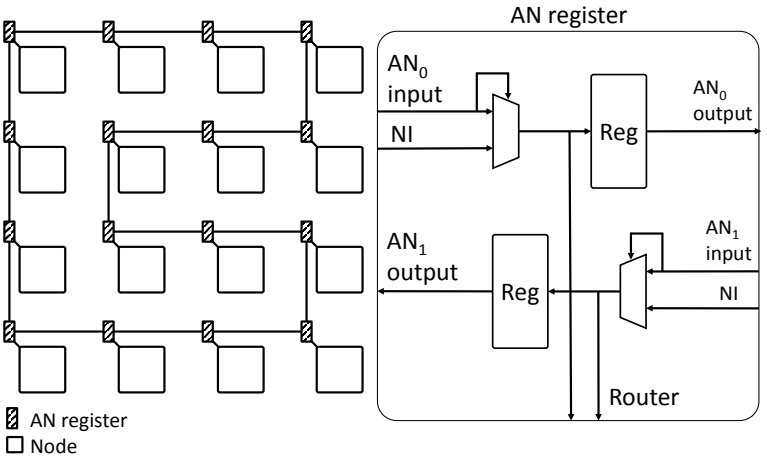


FIGURE 3.56: AN network in a 4x4 mesh.

The logic to support PAPM on the router design is small. Notice that every input port needs to compute whether the port is along the path set between the source and the destination of the ABP message. This logic is provided with the notification source, the destination node which defines the path (source→destination) to be switched on/off and the current node which allows to know whether this node belongs to the path and which ports are involved in it. The outputs of this logic are connected to the counter of each input port, allowing the counter for each buffer to be incremented when needed.

As seen in Figure 3.55, each counter for each port drives the power for each input port. These control signals trigger the actions to power on or off those related buffers.

Switching ports off may cause race conditions since neighbor routers may already have sent flits to the power-gated input port or, at least, already reserved resources to deliver

flits (credits). To avoid these synchronization issues we implemented a simple handshaking protocol similar to the one implemented in the PAPM version for ICARO. A given router A wants to switch off one of its input ports connected to its neighbor B, this protocol simply sends a message from A to B just before switching the port off. Router B replies to this message with an ACK signal in case there are no flits pending to be delivered to A and marks the output port as *disabled* to avoid to be selected in the VA/SA stage in further arbitrations. Otherwise, router B waits until the condition meets to send this ACK. Once the ACK signal is received by router A, the input port is switched off. To switch a port on, a signal is sent to router B to force router B to mark the output port as *enabled*.

3.3.2.2 Activation Network

Figure 3.56 shows the added control network to deliver ABP messages. The network forms a bidirectional ring topology forming a zig-zag structure visiting all the NoC routers and end nodes similar to the dedicated network used in ICARO. For each network hop one simple latch is used and a small multiplexer unit is added. The multiplexer is added on every end-node in order to allow to inject ABP messages. The output of the demultiplexer is wired on every router in order to eject a copy of traveling ABP messages through the new network.

The ABP messages injected through the network will travel along all the ring and will be removed when they reach again the injector end node (one complete lap performed). To do this, each message includes the following fields: source of the path (*src*), destination of the path (*dst*) and a valid bit (*valid*). In order to speedup the ABPs delivery, the ring works at twice the system clock frequency by using latches activated either by rising or falling edge. Since the AN is composed of two rings, each one for one direction, each ABP is duplicated and injected in each direction of the ring. This means that each router will receive each ABP duplicated, thus its affected buffer counters will be increased by 2. To solve this, each DBP received will cause the buffer counter to be decreased by 2. Whenever an ABP message is within the network, the message will get maximum priority to move forward along the ring.

ABP messages are triggered by messages allocation and this may occur up to once per cycle. Due to this and the highest priority of the in-flight notifications, ABP injection may be blocked. Because of this, a small buffer is needed for ABPs storing before injecton. However, we analyzed empirically the required buffer size and we arrived to the conclusion that a 2-slots length buffer is enough to avoid any issue for all simulations performed in Section 3.3.3.2.

3.3.2.3 Power-Down Strategy at End Nodes

Switching off/on buffers incur in power penalties, potentially ruining any power saving achieved by switching them off. In order to amortize this power overhead, the buffer must be powered off a minimum number of cycles (BET). Therefore, one important aspect of PAPM method is deciding when to inject ABP or DBP messages. This is performed at the source end-nodes. To do this, PAPM keeps track of the time between generation of messages for the each destination (TBG_{dst}). At each generated message, the TBG_{dst} value is updated according to Equation 3.4:

$$TBG_{dst} = (TBG_{dst} \times 0.8) + ((T_{current} - T_{last}) \times 0.2) \tag{3.4}$$

where $T_{current}$ represents the current time and T_{last} represents the time where the last message to the same destination was injected.

PAPM implements a bit vector (referred to as *Active Paths Bitmap*) to keep track of the status of all paths. When a message is generated, PAPM checks the status of the path used to reach the destination. If the path is *off*, then an ABP message is injected, changing the state of the bit to *on*.

The message generated is then delivered to the network interface queue and prepared for injection. When the tail of the message reaches the head of the queue (just before injecting it), PAPM checks whether the path has to be powered down or not. The path should be powered off if the time for the next injected message to the same destination is larger than *BET*. This means some power saving will be achieved. Therefore, PAPM enables the DBP bit (which converts the message into a DBP message) in the tail flit of the message if the following condition applies:

$$TBG_{dst} > BET$$

If the expected time between injections is smaller then the path is not switched off.

If the last message sent to a given node did not trigger the DBP bit and no more messages are allocated for that destination node (TBG failed predicting the next allocation time) will cause the path to be kept on indefinitely. To avoid this, a dedicated module (TBG watchdog) is in charge of automatically sending a dedicated DBP message to such destination node after $3 * TBG$ cycles in case of no new message allocation.

TABLE 3.9: General system configuration.

Network configuration	
Topology	8x8 2D mesh
Routing policy	XY
Switching technique	Wormhole (flit-level)
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4 per Virtual Network

3.3.3 Evaluations

3.3.3.1 ICARO-PAPM

In this section we report simulation results obtained for several system configurations. As in previous evaluations, since ICARO's goal is to isolate congested traffic from non-congested ones, the inspected traffic pattern is composed of two components: uniform traffic at low data rate and hotspot traffic consisting in 4 nodes injecting to a single node from time 300 μs to 350 μs (highlighted with vertical blue lines in the figures).

Since ICARO uses an additional VN to isolate congested traffic, all simulations are performed with 2 VNs, each one containing the same number of VCs. The rest of the system configuration parameters are described in Table 3.9. To elaborate different configurations, in order to evaluate our proposal, we set a *baseline* configuration. Starting from this configuration, we modify different parameters (mesh size, router's queue size, number of VCs and number of hotspots) to elaborate a set of benchmarks. All configurations are detailed in Table 3.10.

Since the goal of ICARO-PAPM is to isolate congested traffic in order to keep the background one unaffected, for clarity, latency graphs show results only for background traffic. Note that, latencies for AM/DM messages are also included in the results.

In Figure 3.57, a comparison between ICARO and ICARO-PAPM is performed in order to demonstrate that, to implement a power-gating mechanism, has negligible effects over ICARO. As seen, ICARO-PAPM performs quite similar to ICARO. It is worth to note that, even for configurations in which PAPM messages suffer from high latencies (shown later), there is no significant ICARO-PAPM impact on performance. Regarding power consumption, in Figure 3.58 all results are depicted, showing values for the overall mesh power consumption. As can be seen, before the hotspot is activated, ICARO-PAPM saves up to 35% of power consumption by keeping the *extra-VN* buffers powered off. When hotspot is activated, ICARO-PAPM activates only those buffers that compose the routes needed to deliver congested traffic, therefore, reduces power consumption by 27%. In Figures 3.60 and Figure 3.61, final results for power consumption are shown dividing the power consumption in two sets: when congestion does not arise in the

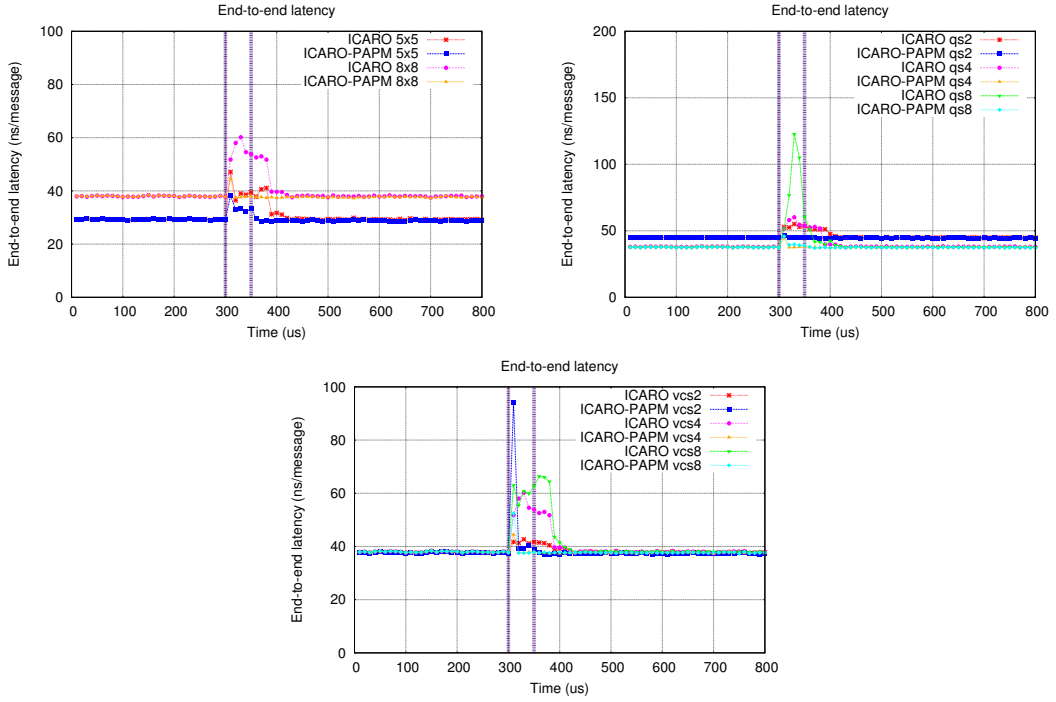


FIGURE 3.57: End-to-end latency comparison between ICARO and ICARO-PAPM for different configuration parameters

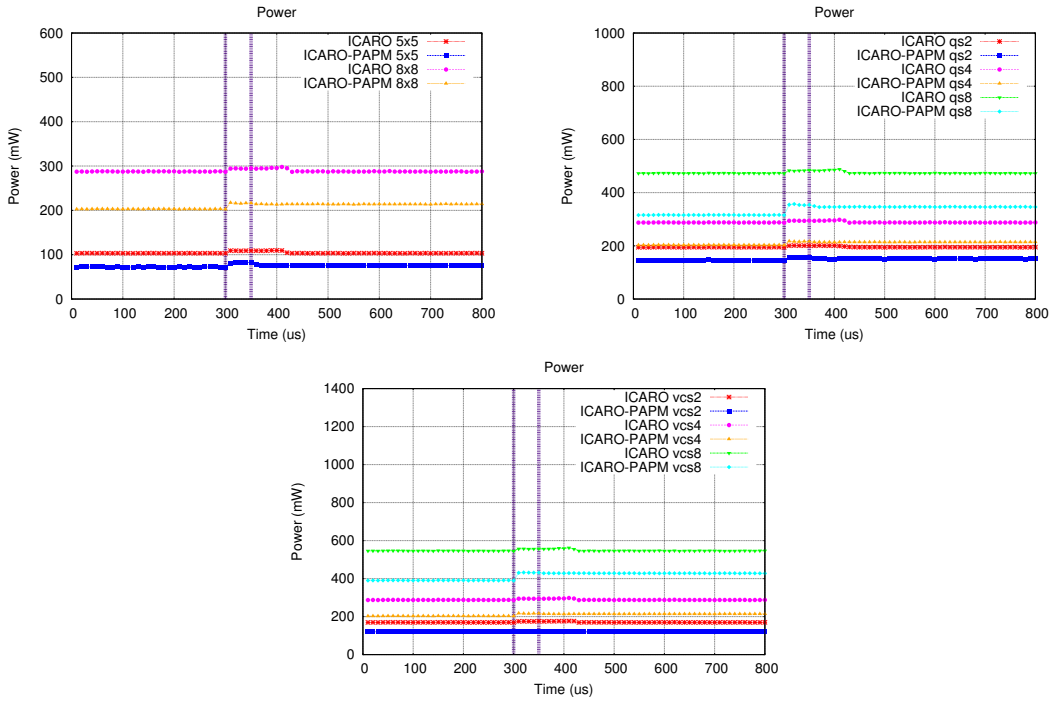


FIGURE 3.58: Power consumption for different configuration parameters

network (hotspot disabled) and when congestion arises in the network (hotspot enabled), respectively. In all the cases ICARO-PAPM succeeds in reducing power consumption significantly.

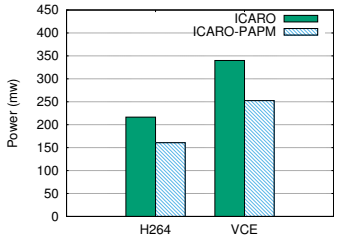


FIGURE 3.59: Average power consumption under realistic multimedia traffic patterns.

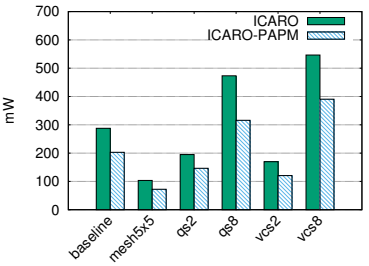


FIGURE 3.60: Average power consumption when no congestion in the network.

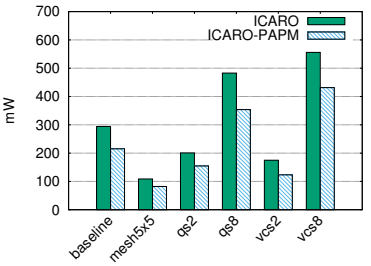


FIGURE 3.61: Average power consumption when congestion traffic in the network.

TABLE 3.10: Scenarios configuration.

Scenarios				
Label	Mesh Size *(nodes)*	Queue Size	Num. VCs	Num. HS
Baseline	8x8	4	4	1
5x5	5x5	4	4	1
qs2	8x8	2	4	1
qs8	8x8	8	4	1
vcs2	8x8	4	2	1
vcs8	8x8	4	8	1

In order to evaluate our proposal under more realistic scenarios we also performed evaluations under more realistic traffic patterns. We followed the same strategy as in [54]. We implement H264 and VCE multimedia codecs realistic traffic patterns and performed simulations with no ICARO and with ICARO-PAPM. Figure 3.59 shows the power consumption for the given traffic patterns without any congestion control and with ICARO-PAPM. As seen, ICARO-PAPM saves power by activating buffers only when necessary, achieving 26% of power saving in both scenarios with no performance loss.

3.3.3.2 PAPM

In this section we evaluate PAPM under different configurations using our in-house NoC simulator. To obtain our power results we used a modified version of Orion v3.0 [57] and Encounter tool from Cadence for calculating power overhead due to the additional logic added by PAPM. Regarding power-on delay, according to the current state-of-the-art [45] we could assume a delay of *0.2ns*. However, in order to show the behavior of our proposal under worse cases, for our experiments we assume a power-on delay of *2ns* (2 cycles).

In order to evaluate our proposal, we perform two analysis following different approaches. First, we simulate a system performing changes of context by changing between different traffic patterns for two different mesh sizes: 4x4 and 8x8. Then, we use realistic

Simulation configuration	
Topology	4x4 2D mesh
Routing policy	XY
Switching technique	Wormhole
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4
Frequency	1GHz

TABLE 3.11: Simulation configuration.

Time (μs)	Pattern	Inj. Rate (f/c)
0-99	uniform	0.1
100-199	bit-reversal	
200-299	bit-complement	
300-399	bit-rotation	
400-499	bit-shuffle	
500-599	transpose	
600-699	tornado	
700-799	butterfly	

TABLE 3.12: Traffic patterns.

traffic [54], increasing its intensity until network saturation to find the upper limit in which buffers are used at their maximum rate while using more realistic traffic.

For our analysis we show results for a system provided with no power-gating mechanism, results for a system implementing PAPM and, additionally we compare also with TooT[44], a recent power-gating proposal described in Section 2.2.2, which essentially switches routers off when traffic with no turns is detected and enables bypasses in the router to keep it in service.

For our first benchmark we perform simulations using several synthetic traffic patterns. Simulation toggles the traffic pattern used along the time in order to emulate a system running different applications. By doing this, we demonstrate that PAPM is able to dynamically adapt the available buffers in the network to fit the application requirements while saving power switching off those which are not used. Details about the traffic patterns used are described in Table 3.12.

One of the main challenges when implementing power-gating based strategies is to hide the delay caused by the powering on process. In addition to this, since our proposal is based in notifications (ABPs) to trigger this process, there is an additional delay caused by ABPs delivery time. However, as seen in Fig. 3.62, the AN is able to deliver all notifications in time thus no significant latency overhead can be appreciated. In the same way, as seen in Fig. 3.64, no impact on throughput can be appreciated. However, as shown in Fig. 3.63, PAPM disables buffers not required to deliver each traffic type, achieving up to 73% and 33% of power savings compared with no power-gating and TooT respectively. It is worth to note that, in addition to overcome TooT saving power, TooT suffers latency overheads while PAPM keeps the latencies unaffected. Similarly, Figures 3.65, 3.66 and 3.67 show the results for a 8x8 mesh network. As shown, the latency when running PAPM increases slightly. This is due to the delay of the AN delivering the ABPs to all nodes in the network. However, this increase is minimal, is lesser than the latency increase using TooT and the power saving of PAPM is still around 65% of improvement compared with the no power-gating case and 21% comparing with TooT.

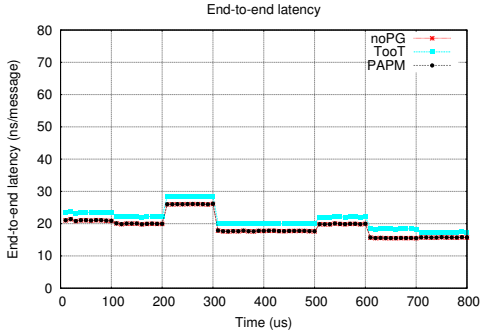


FIGURE 3.62: End-to-end latency.

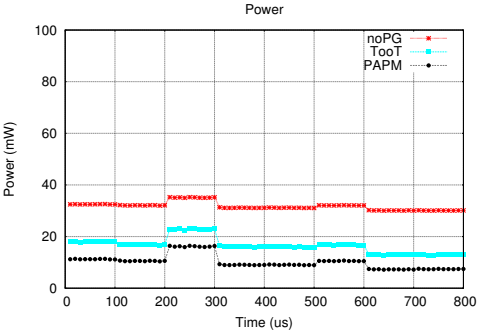


FIGURE 3.63: Power consumption.

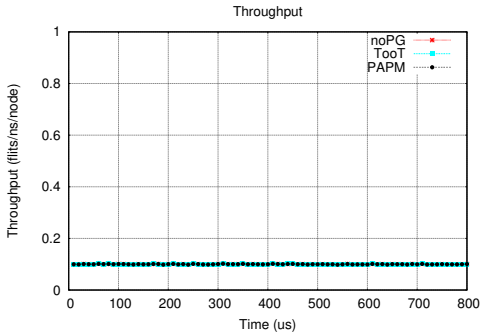


FIGURE 3.64: Throughput.

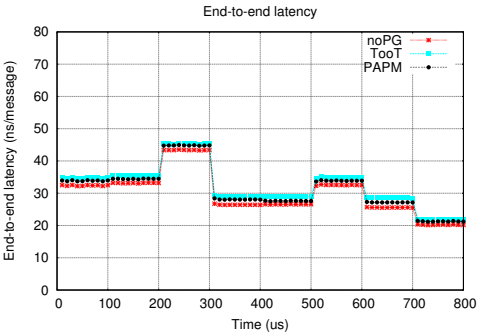


FIGURE 3.65: 8x8 end-to-end latency.

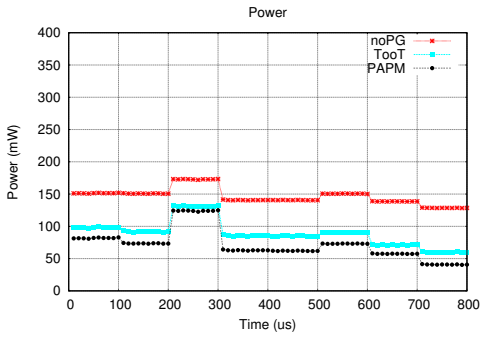


FIGURE 3.66: 8x8 power consumption.

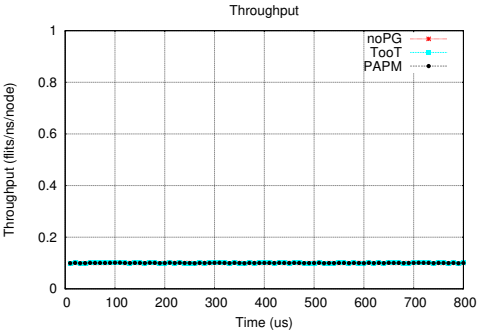


FIGURE 3.67: 8x8 mesh throughput.

For the next analysis a realistic traffic pattern corresponding to the H264 codec is used. To generate this traffic we followed the methodology described in [54]. Since this traffic emulates the H264 video codec traffic, we are able to provide the frame rate at which the codec works. Since flows from any source are sent always to the same destination, the subset of used buffers is always the same. Taking benefit of this, for this analysis we increase the frame rate parameter in order to increase the network load under this traffic in order to analyze how PAPM and TooT react to an increasing network load until saturation using always the same buffers subset. In addition to this, performing this analysis is also useful to evaluate PAPM and TooT under more realistic traffic.

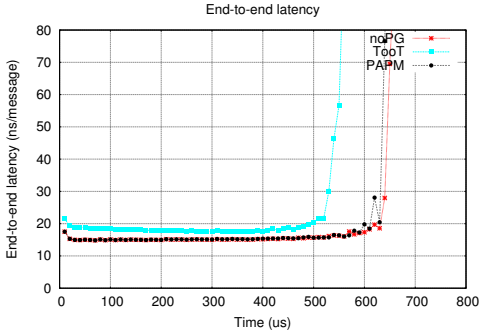


FIGURE 3.68: H264 end-to-end latency.

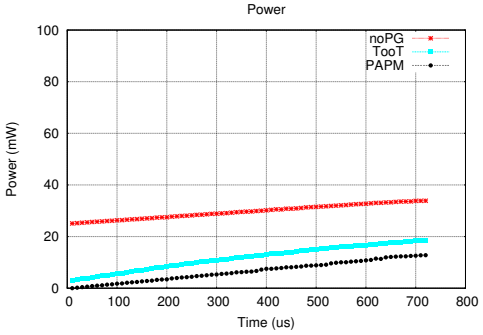


FIGURE 3.69: H264 power consumption.

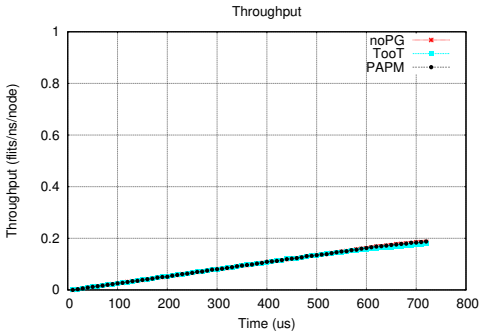


FIGURE 3.70: H264 Throughput.

As seen in Fig. 3.68 and Fig 3.70, PAPM has negligible effect over the latency and throughput, respectively. As seen, at very low frame rate, the network utilization is very low, therefore, while the no power-gating case keeps all the buffers on, PAPM only switches them on when necessary, consuming very low power. Regarding TooT, due to the reduced buffers capacity in routers when bypasses are enabled, causes the saturation to arise earlier, which makes it not suitable under high network load. As long as the frame rate is increased, power consumption for the no power-gating case increases due to the dynamic power component and PAPM reacts by increasing the buffers uptime, increasing the power consumption as well. However, in the worst case (at the highest frame rate), PAPM achieves an average power consumption improvement of 79% compared to the case with no power-gating and of 45% when compared to TooT.

3.4 Proposals Digest

For the sake of understanding, in Table 3.13 we show, for each proposal described in this thesis, key characteristics that differentiate each proposal, to provide a general view of all works and the key differences between them.

TABLE 3.13: Digest of all proposals described in this thesis

	BAHIA	ICARO	ICARO-DVFS	ICARO-DMSD	ICARO-PAPM	PAPM
Detection location	End-node	Router	Router	Router	Router	-
Saturation detection	Data_rate >threshold	>1 sat. queues req. an out_port	>1 sat. queues req. an out_port	>1 sat. queues req. an out_port	>1 sat. queues req. an out_port	-
Detection Implement.	Bit-vector	CP cache	-	-	-	-
Dedicated network	BNN	CNN	CNN (extended)	CaL (CNN+lats.)	CNN	AN
Network Topology	All-to-all	Segmented ring	Segmented ring	Segmented ring	Segmented ring	Segmented ring
Data sent	*HIGH/LOW*	Congested points	Cong. points, V&F reqs.	Cong. points, latencies	Cong. points, AM msgs.	ABP/DBP messages
Compared to	no-BAHIA	no-ICARO, FVADA/AVADA	DVFS	DMSD	ICARO	TooT

Chapter 4

Head-of-Line Blocking Avoidance in Networks-On-Chip

- **Authors:** José Vicente Escamilla (Universitat Politècnica de València), José Flich (Universitat Politècnica de València) and Pedro J. García (Universidad de Castilla-La Mancha)
- **Type:** Conference
- **Conference:** 3rd Workshop on Communication Architecture for Scalable Systems (CASS)
- **Location:** Boston, MA, USA
- **Year:** 2013
- **DOI:** 10.1109/IPDPSW.2013.214
- **URL:** http://ieeexplore.ieee.org/document/6650958/
- **Citation:** [59]

4.1 Abstract

Many-core chip designs are the current manufacturing trend for high-performance computing. Different challenges lead to different designs, whether general purpose-driven chip multiprocessors (CMPs) or application-specific multiprocessor system-on-chips (MPSoCs) are deployed. An emerging problem is on-chip network congestion, due either to several traffic flows requesting the same resources (e.g. memory controllers) or to bursty traffic interfering with other flows.

In this paper we propose BAHIA, which enables dynamic separation of bursty traffic from non-bursty one, thereby removing all the contention effects of bursts with a minimal impact on network overhead and with a marginal increase in area requirements. Results demonstrate a robust and effective splitting of traffic, so that non-bursty traffic achieves the desired low latencies even in bursty-prone conditions. By contrast, in our experiments, when BAHIA is not used, non-bursty traffic gets congested, latency increasing significantly and, in some cases, reaching a factor increase of 3x.

4.2 Introduction

The high-performance computing domain is taking advantage of the inclusion of multicore solutions in the form of Chip Multiprocessor (CMP) and System-on-Chip (MPSoC) systems. As the integration scale goes further, more cores, nodes, or processing units are included in the same chip. Examples of the many-core integrated Intel CMPs are: The Xeon Phi coprocessor [60] with 60 cores, and the single chip cloud computer (SCC) [61] with 48 cores. The Tile-Gx [7] from Tilera, with up to 72 cores, represents a good example for a high-end MPSoC system. These systems provide support for the specific needs of the different targeted applications such as multimedia, wireless networking, and cloud-computing.

Both design platforms, CMPs and MPSoCs, rely on an interconnection network infrastructure that provides the communication between all the processing nodes. This must be a high-bandwidth, low-latency network to avoid slowing down processors while waiting for remote data. Networks-on-chip (NoCs) suit well when a large number of processing nodes are present [6], as is the case of the Intel prototypes and Tilera products. In general, NoC design is challenging due to the tight constraints found in on-chip systems. Thus, an NoC must be simple in its mechanisms, exhibiting low hardware overhead, low power-demanding, and at the same time being performance-efficient, independently of the applications running on the system.

As technology advances, we will have more complexity in the chip, leading to more devices. Also, specialization will drive the inclusion of dedicated devices as accelerators,

encoders, DMA devices, etc. This heterogeneity and the increasing number of components will drive a change in the traffic present in those devices. We can expect traffic bursts flowing from device to device, at intermittent and unpredictable frequencies. This kind of traffic may create temporary oversubscribed ports (hotspots) where the traffic is concentrated, thereby leading to the appearance of network congestion that is likely to have a negative impact on the rest of the traffic. Specifically, in congestion situations, the Head-of-Line (HoL) blocking phenomenon [14][62] is likely to appear, that consists in congested traffic slowing down other traffic flows throughout the network. As we show in this paper, efficiently dealing with the problems derived from congestion can significantly improve the overall chip performance.

Indeed, in this paper we present BAHIA (Burst-Aware HoL-blocking Injection Avoidance), a mechanism that dynamically detects bursty traffic in the network, then isolating the burst and thus guaranteeing that non-bursty traffic is unaffected. The BAHIA method is targeted on lightweight NoC designs, where transmission latency is of outmost importance, thus, no additional mechanisms built inside the network are added to the switches (i.e. congestion control mechanisms, traffic separation). Indeed, we follow the same approach as in [63], so moving the complexity to the network interfaces (NIs). Notice that BAHIA can be complemented with any sophisticated mechanism built inside the network. Indeed, BAHIA is implemented at the end-nodes (NIs) and thus, it is orthogonal and complementary to NoC mechanisms.

The rest of this paper is organized as follows. Section 4.3 presents the related work. In Section 4.4 we thoroughly describe the key aspects of the BAHIA mechanism. Next, in Section 4.5, we detail the evaluation scenarios and the results obtained, and finally in Section 4.6, we present some conclusions and future work.

4.3 Related work

There is a number of solutions in [64], [65], and [66] focused on reducing the negative effects of resource sharing through quality of service (QoS) policies and mechanisms, based on priority schemes. Although all these solutions can alleviate or delay the negative impact of network congestion by prioritizing different traffic types, their main objective is to differentiate traffic and they are not actually focused on dealing with congestion by itself. As a consequence, congestion may appear within each traffic class due to unpredictable traffic patterns.

In [67] authors first make an analysis of the impact of resource sharing with different traffic patterns and the implication of dependencies between packets of the same data flow in the efficient utilization of these resources. Finally authors propose to change the abstraction unit from mapping packets to virtual channels to mapping flows to virtual channels following a destination-based mapping policy.

Regarding specific congestion-control mechanisms, the authors in [16] present a mechanism for congestion control and analyze its relation to the system scalability for bufferless on-chip networks. As the authors describe, their experiments show congestion problems in this kind of networks (system size ranges from 16 to 4096 nodes), thus demonstrating the need for a congestion control mechanism to prevent performance degradation. However, as this work focuses on buffer-less networks, the results cannot be directly applied to the buffered NoCs we consider in this paper. The main corpus of congestion-control solutions on buffered NoCs are based on the idea of congestion-awareness mechanisms implemented on the switches, either based on deterministic or adaptive routing. The solutions presented by the authors in [20], [68], [69], and [70], describe mechanisms that collect congestion information from the neighbor nodes through the routing process and ingress/egress buffer monitoring, in order to offer an alternative path to route around a congested area. However, this strategy may end up producing more congested resources, as it is impossible to avoid the congested region if all the congested traffic has the same target (e.g. the memory controller).

In the case of high-end MPSoCs, all the previous congestion methods can be effectively used as well. However, these methods do not specially deal with traffic bursts. Bursty traffic has been analyzed and explored within the concept of QoS especially when addressing how QoS application requirements (latency, jitter, and bandwidth) can be met within the network. Here we are not interested in the QoS aspect of the traffic, but on the congestion effects that uncontrolled bursts may create. Indeed, in [71] and [72] the importance of bursty traffic in the congestion control framework is pointed out. In that sense, the BAHIA method described in the next section addresses the negative effects of bursty traffic.

In [73] the problems derived from bursty traffic are addressed by increasing buffer size in order to get room enough to absorb bursts. This approach is relatively expensive in terms of silicon area and power, and as reported in the article, with non-bursty traffic this results in a suboptimal utilization of the resources. In addition, it is difficult to predict burst sizes, hence probably a burst may overflow buffers, then resulting in contention. In BAHIA, however, buffer size is not modified as bursty traffic is separated at the sources.

In [74] a solution to reduce the latency in worst-case bursty traffic is proposed. However, this mechanism is based on temporarily ejecting packets and later re-injecting them with a priority-based approach. This achieves good results as it helps in the worst case, but at the cost of increasing latency of newer packets. Moreover, if we consider a scenario with several virtual networks, this mechanism requires three queues per virtual network at each node, thus becoming an expensive solution in an NoC context.

In [75] authors propose a mechanism for improving substantially the deflection rate in bufferless NoCs by implementing an end-to-end flow control technique. This mechanism is a credit-based algorithm that keeps a clumsy record of the end-node buffer availability

instead of the intermediate buffers in order to throttle the injection data rate. Authors propose a clumsy credit-based counter since, thanks to the deflection mechanism, an strict counter is not necessary. This mechanism has some similarities with BAHIA in the sense that an end-to-end load parameter is measured in order to take a decision to eventually improve the network efficiency. However, as mentioned above, BAHIA only works for buffered networks-on-chip and the proposal in [75] makes no sense in such kind of networks.

On the other hand, many related solutions have been proposed in the off-chip context (for clusters or Massive Parallel Processors) that notify (end-to-end) congestion to the sources so that they cease or reduce packet injection (e.g. the InfiniBand congestion-control mechanism [76]). The main difference with BAHIA is the strict minimized implementation overhead pursued when designing BAHIA, which is a requirement in the context of NoCs.

4.4 BAHIA Description

BAHIA (Burst Aware HoL-blocking Injection Avoidance) provides a method to isolate, at runtime, detected bursty traffic in a network, in order to prevent bursts from causing HoL-blocking. Detection of bursty traffic is performed at any end-node receiving a burst. All the end-nodes are then notified of this detection, so that thereafter the bursty traffic can be identified in order to be separated from non-bursty traffic, thereby avoiding the HoL-blocking that the former could produce to the latter. BAHIA makes use of virtual networks to separate traffic, hence BAHIA requires at least two virtual networks: the "default" virtual network (hereafter default-VN), for non-bursty traffic, and an "extra" virtual network (hereafter extra-VN) for bursty traffic. Therefore, if no bursty traffic is detected, the traffic is injected always through the default-VN, but when a burst is detected, the bursty flow is mapped to the extra-VN. It is worth mentioning that, although BAHIA makes a special use of virtual networks, it supports virtual channels implementation over such virtual networks.

4.4.0.1 Burst Detection

As mentioned above, the detection of bursty traffic is performed at the end-node receiving the burst. For that purpose, each end-node periodically calculates its rate of received traffic. The traffic rate is calculated every "polling interval" (PI) cycles, which is a predefined parameter of BAHIA. If that rate exceeds a given high-threshold (HT) value, this end-node will notify the other end-nodes that it is receiving bursty traffic. Similarly, any end-node notifying bursty traffic must be able to detect the end of

the burst. For that purpose, the traffic reception rate is compared with a given low-threshold (LT) value. Accordingly, in this case, all the end-nodes will be notified about the end of the burst. It is worth mentioning that an appropriate configuration of the two aforementioned thresholds (upper and lower) is important to achieve the best BAHIA performance. Indeed, in our evaluation experiments we have thoroughly tuned these values, as explained in Section 4.5.

On the other hand, an alternative burst detection mechanism could be conceived at the sender. Indeed, if a node is going to inject a burst then it can know that in advance. However, detecting at the senders cannot guarantee the effective detection of different lightweight bursts from different sources to the same destination, thereby being not so efficient in preventing the negative effects of aggregate bursts. Hence, we opted for performing burst detection at the receiver part of the end-nodes.

4.4.0.2 Burst Notification

In order to notify a traffic burst to all the end-nodes, BAHIA makes use of a simple dedicated signaling network (Burst Notification Network, BNN). Basically, the whole BNN is a set of one-bit-wide overlapped control networks, each one managed by a specific end-node. Each one-bit control network connects its manager end-node to the rest of end-nodes, the former being the only one able to activate/deactivate the signal (i.e. to set the bit to one/zero) of this control network, while the latter being just signal receivers. Thus, every end-node owns an exclusive one-bit-wide control network to notify bursts events to the rest of end-nodes. Figure 4.1 shows the one-bit-wide control network managed by end-node 0, but note that every end-node owns a similar control network, so that in this 4x4 mesh network, there would be other 15 one-bit-wide control networks besides the one shown. The overlapping of these control networks allows every end-node to notify bursts without risk of collisions with notifications from other end-nodes. Therefore, the BNN can be viewed as an N-bit-link, where N is the number of end-nodes and every bit (wire) in the link corresponds to a specific end-node in the system.

Any end-node notifies the rest of end-nodes of a burst by setting to high value the signal of its BNN line. Due to the simplicity of that signal, it reaches all the end-nodes in a few cycles. The time spent in propagating and processing this signal is modeled by the "notification delay" (ND) parameter in the simulations. It is worth mentioning that the processing of this signal is negligible (and so the required hardware). Regarding the area overhead introduced by the BNN, in [48] an additional dual-network for routing data transmission is proposed. This dual-network comprises the logic needed for coding and decoding data, flow controlling, handling transmission failures mechanisms, etc. Despite of its relative complexity, the authors of the referred paper conclude that the area overhead of the hardware required to implement their proposal is 12.5% of the switch. Note that the BNN consists just in a set of wires which are set to a high or

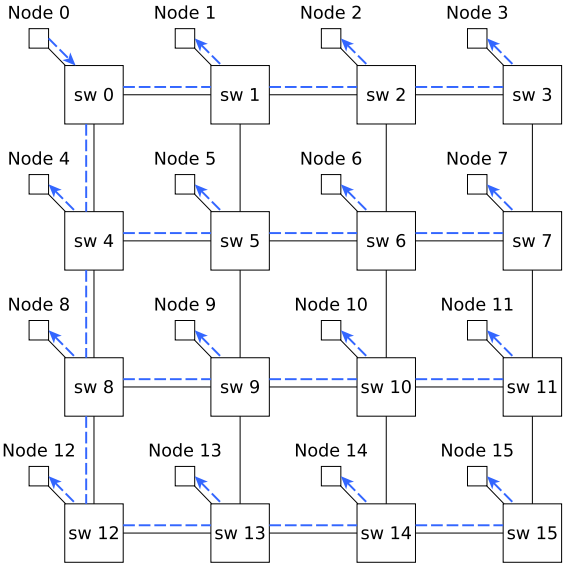


FIGURE 4.1: Node 0 communicates burst events through this 1-bit network.

low signal value for notifying, hence no logic for processing data is needed. Therefore, relying on the hardware overhead study performed in [48], we can conclude that the hardware overhead for implementing the BNN is negligible.

4.4.0.3 Traffic Separation

Every end-node implements a "burstiness bit-vector", each bit corresponding to a specific end-node in the network, as can be seen in Figure 4.2. When an end-node detects a high signal value in the BNN line associated to an end-node, the former will set to one in its burstiness bit-vector the bit corresponding to the latter. Once generated, all the messages are mapped to the default-VN, thus they are initially stored in the queue associated to that virtual network (default-VN queue). At every clock cycle the head of all default-VN queues are checked searching for the head of a message: if one is found, its destination is checked to obtain the value of the corresponding bit in the burstiness bit-vector. If that bit is set to zero, the message remains mapped to the default-VN; otherwise, the whole message is transferred to the extra-VN. These checking&transferring processes can be executed while reading from the queue currently selected for injecting, hence a message at the head of a queue can be transferred to the extra-VN queue while the next one is injected (if it must not be transferred to the extra-VN too). It is worth mentioning that in a real hardware implementation the burstiness bit-vector could be replaced by simply inspecting the signals in the BNN lines.

In extreme scenarios with several bursts addressed to many end-nodes, and with large burst duration, the extra-VN queues may get full. In these cases messages cannot

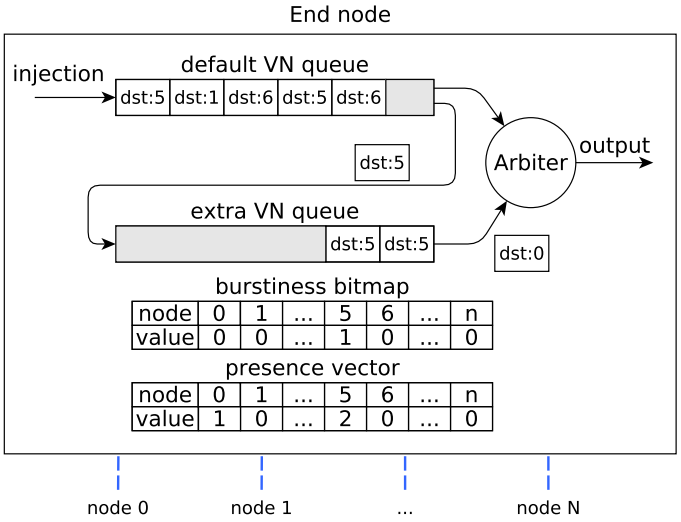


FIGURE 4.2: Forwarding of messages in a source.

be transferred from the default-VN queues to the extra-VN ones, thus the default-VN queues with messages at their head waiting to be transferred to the extra-VN queue will get blocked until messages at the extra-VN queue are drained. Note that it is an extremely unlikely case: indeed, when simulations were performed with realistic benchmarks this never happened.

Figure 4.2 shows the basic structure of the sender part of an end-node that has messages addressed to end-nodes 1, 5 and 6. The message at the header is addressed to end-node 5. The bit corresponding to end-node 5 in the burstiness bit-vector is set to one (i.e. end-node 5 previously notified that it was receiving a burst), so this message must be mapped to the extra-VN. Indeed, before injection, the arbiter at the sender node checks the burstiness bit-vector and transfers the message addressed to end-node 5 to the extra-VN queue, so that this message will be later injected from that queue.

It is worth pointing out that, although messages are injected either from the default-VN queue or from the extra one, all of them are initially mapped to the default-VN. This is because, if an end-node directly maps to the extra-VN the messages addressed to an end-node that has recently notified a burst, there may be messages still stored in the default-VN queue that are addressed to the same destination, and this could introduce out-of-order message injection (and so delivery) as queue selection policy is based on a simple round-robin algorithm. Hence, to guarantee in-order message injection and delivery, all the messages are first mapped to the default-VN and the arbiter is provided with some additional intelligence to check the burstiness bit-vector, in order to evaluate whether a message should be directly injected from the default-VN queue or it should be transferred (by changing pointers) to the extra-VN.

Once an end-node notifies that it is no longer receiving bursty traffic (by setting to low value the signal of its BNN line), the other end-nodes will reset the corresponding bit in their burstiness bit-vector. Thereafter, new messages addressed to this end-node will be injected from the default-VN queue. However, this may also introduce out-of-order message injection and delivery, as messages addressed to the end-node may remain in the extra-VN queue. The example of Figure 4.2 also shows a situation where there are messages in the extra-VN queue addressed to an end-node (specifically, end-node 0) whose associated bit in the burstiness bit-vector has changed from one to zero.

In these cases, in-order packet delivery can be preserved if messages addressed to a specific end-node are injected from the default-VN queue only if there are no messages addressed to the same destination in the extra-VN queue; otherwise, the packet must be transferred from the default-VN queue to the extra-VN one. However, this makes necessary other information than that of the burstiness bit-vector, besides some additional tasks for the arbiter. Specifically, every end-node in BAHIA implements a presence vector that contains an element per end-node in the network, and every element is a counter indicating how many messages addressed to this end-node are stored in the extra-VN queue. Every counter is incremented each time a message addressed to the corresponding end-node is moved from the default-VN to the extra-VN one, and decremented when messages are injected towards that end-node from the extra-VN. When a message reaches the head of the default-VN queue and the bit associated with its destination in the burstiness bit-vector is set to zero, the counter associated with that destination in the presence vector is also inspected: if the value of that counter is zero, the message is injected from the default-VN queue; otherwise, the message is moved to the extra-VN queue.

Note that, although BAHIA has been evaluated in this work assuming a deterministic, dimension-order (XY) routing algorithm, it is suitable to any deterministic or adaptive routing algorithm. However, in-order message delivery is only granted using deterministic routing.

The whole mechanism necessary to keep in-order message injection is a post-processing mechanism, in the sense that messages are mapped to their final virtual network once they reach the head of the default-VN queue, and not before. As mentioned above, the arbiter should be in charge of performing this post-processing mechanism, that can be summarized in the next pseudocode:

Note that post-processing can be performed in parallel to the transmission of a message, either from the default-VN queue or from the extra-VN queue.

Once the end-node injects a packet, the packet will advance towards its destination through the same virtual network. That is, if the packet is injected through the default-VN, it will advance through the devault-VN located on every switch. If, on the contrary,

```
for each default vn in the end-node do
    if !isVNempty(vn) then
        if isNodeReceivingBurst(msg.destination) ||
          numFlitsInExtraVN(msg.destination) > 0 then
            moveMessageToExtraVN(msg);
        end
    end
end
```

the packet is injected from the extra-VN, it will advance through the extra-VN located on every switch. Thus, every switch will implement two queues at minimum to support BAHIA.

4.5 Evaluation

In this section we present an evaluation of BAHIA based on the results of simulation experiments performed in bursty traffic scenarios. First, we describe the simulation environment. Next, we show several analyses of BAHIA performance based either on results obtained with only synthetic traffic or on results obtained with realistic traffic patterns together with background synthetic traffic. Specifically, we firstly offer a robustness analysis of the different parameters that define the BAHIA behavior, carried out establishing a baseline configuration, then running simulations with different values of each parameter, in order to find out the optimal values to configure these parameters. Next, we compare the results obtained with and without BAHIA when using the minimum number of virtual networks. Finally, we perform an analysis of the influence of the number of virtual networks over BAHIA and the performance improvement regarding similar no-BAHIA scenarios. For all these analyses, keep in mind that we do not target overall throughput increase or overall latency reduction. Our aim is to separate bursty traffic from non-bursty one, thereby keeping the latter unaffected by the HoL-blocking the former may produce.

4.5.1 Simulation Environment

An in-house NoC simulator has been used for the experiments. Results are shown every 5000 simulated cycles. Table 4.1 summarizes the configuration of the scenarios modeled in our experiments. Specifically, the network topology modeled in all the experiments is a 2D mesh built from 16 switches arranged in a 4x4 mesh distribution, each switch being connected to a single end-node. Regarding traffic patterns, for the first analysis in Section 4.5.3.2 an only-synthetic-traffic pattern is used. This pattern (traffic pattern A in Table 4.1) consists of background uniform-traffic injected at a rate of 0.2 flits/cycle/node,

together with 2 bursts created by 4 hotspots following a 4-to-1 strategy where nodes inject at 1 flit/cycle/node during 50000 cycles. For the rest of analyses we use traffic pattern B in Table 4.1, which consists of realistic traffic generated from extrapolated MCSL traces from the H264 video encoder [50], together with background synthetic traffic generated by all nodes injecting at a data rate of 0.3 flits/cycle/node with a uniform distribution of destinations. The latter traffic is used as our goal is to avoid the HoL-blocking that traffic bursts derived from the video encoder processes may cause to other traffic. Regarding the number of virtual networks we consider scenarios with 2, 4, or 8 virtual networks. For the BAHIA case we keep a single extra-VN and the remaining virtual networks are used as normal (defaults-VN) queues, so that all the bursty traffic is mapped into the same virtual network. For the no-BAHIA case all the queues are used equally.

Topology	4x4 2D regular mesh	
Virtual networks	no BAHIA	2, 4 or 8 VNs
	BAHIA	1,3 or 7 default-VN(s) + 1 extra-VN
Routing policy	XY	
Switching technique	Wormhole	
Flow control	Stop&go	
Flit size	4 bytes	
Message size	10 flits	
Switch queue size	16 flits	
Traffic patterns	A	uniform 0.2 flits/cycle/node + 4 hotspots 1 flit/cycle/node
	B	h264 video enc. + uniform 0.3 flits/cycle/node

TABLE 4.1: Scenario configuration for bursty traffic.

4.5.2 Parameters Tuning

BAHIA behavior is defined by four parameters: BNN notification delay (ND), high-threshold (HT), low-threshold (LT) and polling interval (PI). In order to explore the robustness of the mechanism, we have carried out an analysis of every parameter by independently simulating different variations of the parameters.

In all the experiments of this analysis traffic pattern B in Table 4.1 has been used, and 2 virtual networks (1 default-VN + 1 extra-VN) are assumed. The baseline configuration, and the different values of the parameters used for the different simulations are shown in Table 4.2.

First, we analyze the effect when varying the notification delay of the BNN network. In Figure 4.3 we can see the results. This figure shows how notification delay has negligible effects on the BAHIA behavior The figure shows the latency of messages (only for the synthetic-traffic part of the traffic pattern) when the BAHIA mechanism is running

	HT (flits/cycle)	LT (flits/cycle)	PI (cycles)	ND (cycles)
baseline	0.45	0.35	1000	2
HT analysis	0.4, 0.45, 0.6, 0.75, 0.9	0.35	1000	2
LT analysis	0.45	0.25, 0.35, 0.4	1000	2
PI analysis	0.45	0.35	100, 200, 400, 800, 1600	2
ND analysis	0.45	0.35	1000	1, 2, 4, 8, 16

TABLE 4.2: BAHIA robustness analysis configuration.

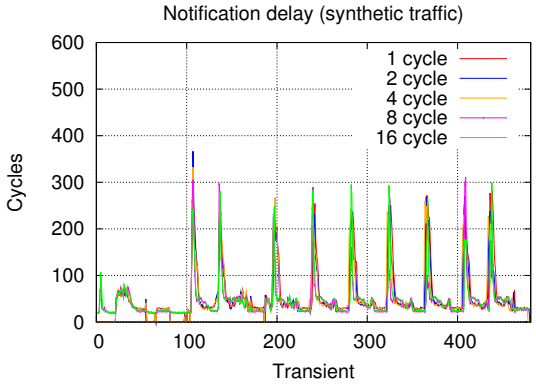


FIGURE 4.3: Notification delay (ND) analysis.

for different notification delays, from 1 cycle up to 16 cycles. As can be seen, latency of messages is unaffected and show almost the same values. This gives some reliability against unexpected jitter delays and grants flexibility for hardware implementation, since BAHIA has no strict delay requirements.

Next we analyze the effect when varying the value of the detection threshold (high-threshold, HT, at the receiver side of the end-nodes). Figure 4.4 shows results for different values of the HT, ranging from 40% to 90% of traffic reception rate. Note that, the lower the threshold, the more aggressive the mechanism (i.e. more sensitive to traffic bursts), but it may incur in false positives (i.e. non-bursty traffic detected as bursty). By contrast, the higher the threshold, the more selective the mechanism, thus it may not detect lightweight bursty traffic. As we can see in Figure 4.4, all the considered values of HT except the lowest one (40%) produce similar results. Thus we can conclude that this parameter should be configured so that BAHIA is not too sensitive, i.e. the HT value is high enough to avoid false positives.

Similarly, Figure 4.5 shows the results obtained when varying the value of the low-threshold (used to detect the end of bursts). As can be seen, this parameter does not affect BAHIA performance because, when a burst ends, all virtual networks are free from HoL-blocking effects, so traffic flows smoothly through all virtual networks, thus no matter which virtual network messages are mapped to.

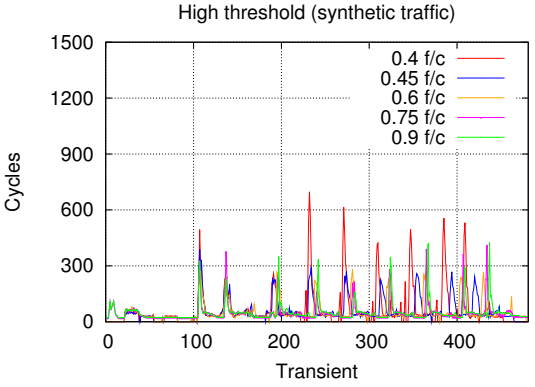


FIGURE 4.4: High-threshold (HT) analysis. Average flit latency.

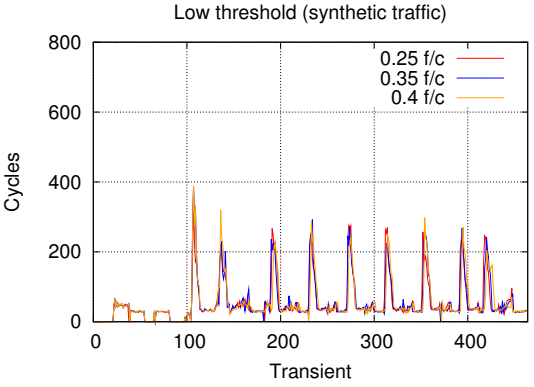


FIGURE 4.5: Low-threshold (LT) analysis. Average flit latency.

Finally, the value of the polling interval has been tested. Note that the polling interval tuning presents a close relationship with the high-threshold analysis: having a small polling interval may be similar to having a lower HT value. The contrary also applies: a large polling interval could lead to the mechanism filtering short transient bursts. As can be seen in Figure 4.6, for all the considered values we get a similar behavior

We conclude, then, that the smallest polling interval among those considered is convenient and not harmful.

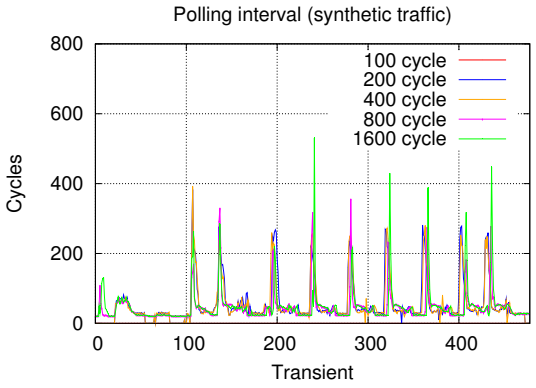


FIGURE 4.6: Polling interval (PI) analysis.

Summing up, any combination of the considered values for these parameters, except the lowest value for HT, would produce similar results. Thus, hereafter we assume that an appropriate configuration of parameters can be ND=4 cycles, HT=0.6 flits/cycle, LT=0.4 flits/cycle, PI=400 cycles, hence such configuration will be used for the next analyses.

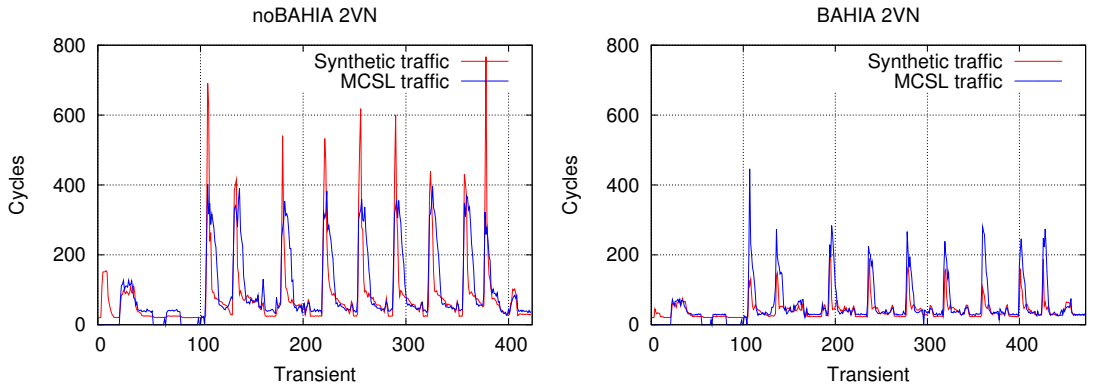


FIGURE 4.7: Latency for BAHIA and no-BAHIA, 2 VNs, traffic pattern B.

4.5.3 BAHIA vs no-BAHIA Analysis

In this section we carry out a general analysis of BAHIA in comparison with similar scenarios without BAHIA in order to quantify the capability of BAHIA to separate bursty traffic from non-bursty one. In the previous section we have delimited a set of appropriate values for the parameters that define BAHIA behavior for the scenario used in the simulations, so in this analysis we have set BAHIA parameters according to these values.

Average latency (cycles)			
	noBAHIA	BAHIA	Improvement
2VN synthetic traffic	56.78	40.40	40.54%
2VN MCSL traffic	130.02	121.89	6.67%
4VN synthetic traffic	85.89	39.31	118.47%
4VN MCSL traffic	240.10	102.40	134.46%
8VN synthetic traffic	133.71	37.65	255.15%
8VN MCSL traffic	464.21	110.88	318.66%

TABLE 4.3: Average latency for BAHIA and no-BAHIA scenarios.

4.5.3.1 Simplest Configuration Analysis

For this analysis, we have performed simulations for the simplest configuration in terms of virtual networks, i.e. 2 virtual networks (as BAHIA requires a minimum of 1 default-VN and 1 extra-VN). Figure 4.7 shows the latency achieved with and without BAHIA for traffic pattern B in Table 4.1. Note that the results for the synthetic and MCSL components of the traffic pattern are shown separately, as different series in the same graph. Average numeric results can be seen in the first two entries of Table 4.3. As can be seen in Figure 4.7 BAHIA achieves a moderated improvement of approximately 40% in the average latency with respect to the no-BAHIA case. Indeed, as can be seen

in Table 4.3, BAHIA achieves a moderated improvement of approximately 40% in the average latency with respect to the no-BAHIA case.

4.5.3.2 Number of Virtual Networks Analysis

Now, we turn our attention to scenarios where different numbers of virtual networks are available. Specifically, we consider the cases of 2, 4, and 8 virtual networks. As mentioned above, for the BAHIA case we keep a single extra-VN, the remaining virtual networks being used for non-bursty traffic. For the no-BAHIA case all the queues are used equally. For this analysis we have used traffic patterns A and B in Table 4.1, in order to get a more complete comparison between BAHIA and no-BAHIA cases.

We firstly analyze results when traffic pattern A in Table 4.1 (i.e. only synthetic traffic) is used. Assuming this traffic pattern, Figure 4.8 shows results of latency along time for a network without BAHIA. The results correspond to the overall average latency for all VNs. Clearly we can see a latency increase when contention exists due to the appearance of hotspots, regardless the number of virtual networks. In Figure 4.9 the overall average latency for all default-VNs with BAHIA is shown. It can be seen that when traffic bursts starts, BAHIA quickly detects an excessive received data rate, thus it is notified to the rest of end-nodes and such traffic is isolated in the extra-VNs, keeping the default-VNs latency low. On the other hand, as can be seen in Figure 4.10 the latency of the extra-VNs increases as bursts are mapped to them. Note that these BAHIA results do not significantly vary with the number of virtual networks, and they show clearly how BAHIA isolates traffic-bursts into the extra-VN.

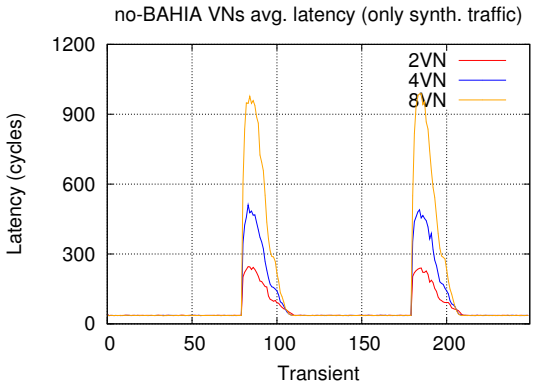


FIGURE 4.8: Overall average latency without BAHIA, traffic pattern A.

Secondly we focus on the results obtained for traffic pattern B in Table 4.1 (i.e. realistic MCSL traces together with synthetic background traffic), with and without BAHIA. Note that the results corresponding to the synthetic part of traffic pattern B are shown separately (in Figure 4.11) from those corresponding to the realistic part (MCSL traces) of this pattern, that are shown in Figure 4.12. Average numeric results can be seen in

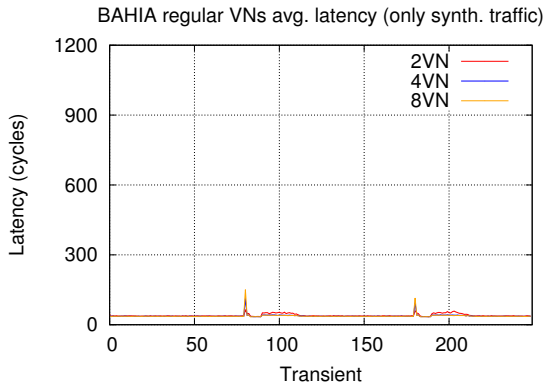


FIGURE 4.9: Default-VNs average latency with BAHIA, traffic pattern A.

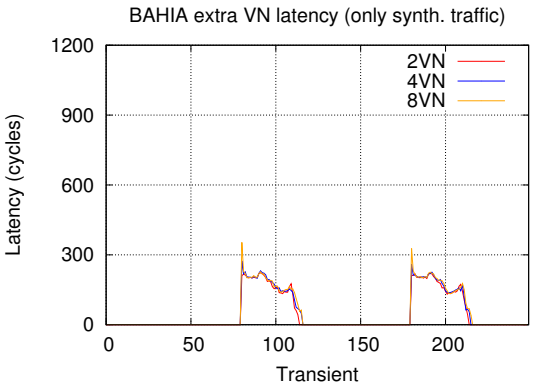
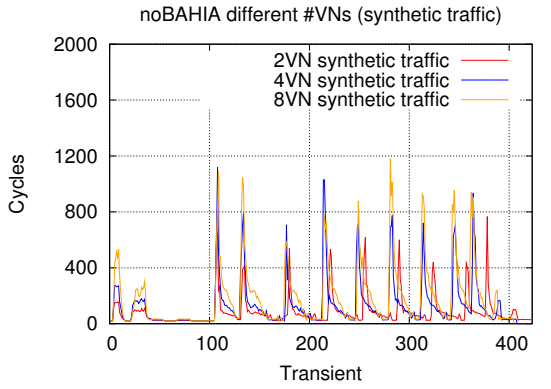


FIGURE 4.10: Extra-VN average latency with BAHIA, traffic pattern A.



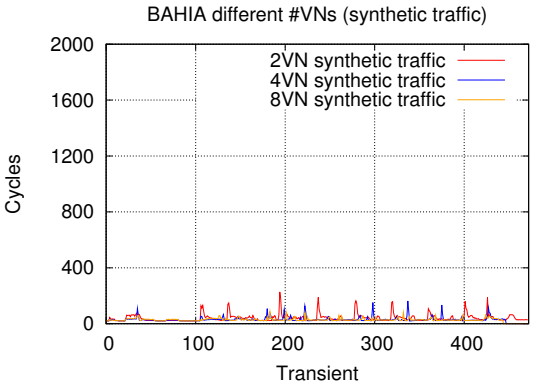


FIGURE 4.11: Latency for the synthetic part of traffic pattern B, without and with BAHIA.

Table 4.3. As can be seen in these figures, the higher the number of virtual networks, the higher the latency increase in the no-BAHIA case. By contrast, when using BAHIA, non-bursty traffic latency keeps bounded and minimized, reaching high latency reductions up to a factor of 3x.

In order to better appreciate the average latency improvement of BAHIA in relation to the no-BAHIA scenarios with different number of VNs in Figure 4.13 we can see the average latency for such cases. Clearly, without BAHIA latency increments linearly with the number of VNs while the scenarios implementing BAHIA keep it constant.

4.6 Conclusions and Future Work

It is well-known HoL-blocking that considerably degrades the overall network performance. In fact, this harmful effect is pronounced as more and more virtual networks are available in the system. In this paper we have presented a solution (BAHIA) to solve this problem by isolating bursty traffic from non-bursty one. With this mechanism

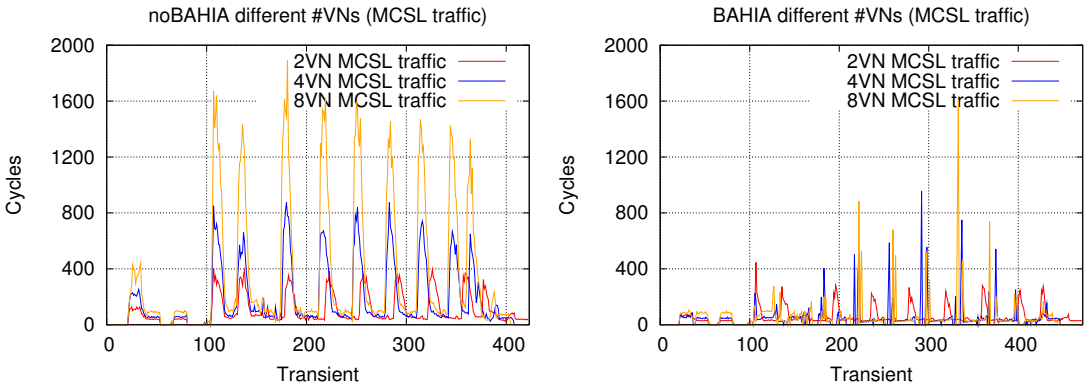


FIGURE 4.12: Latency for the MCSL part of traffic pattern B, without and with BAHIA.

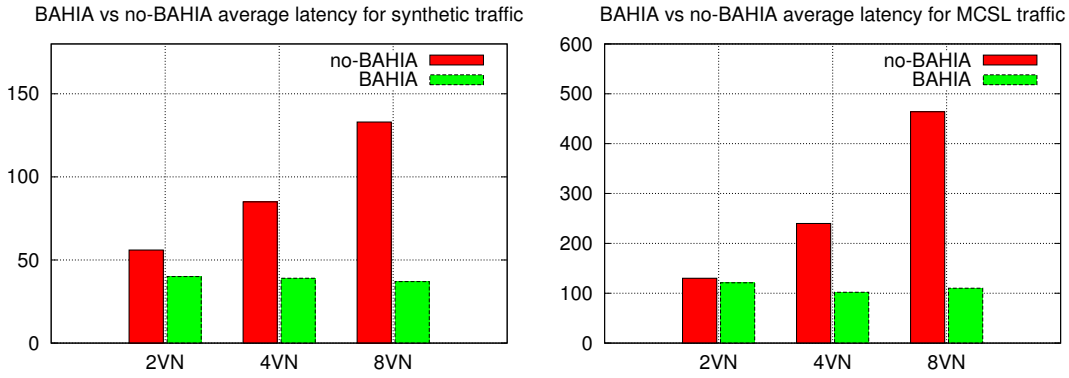


FIGURE 4.13: Average latency without and with BAHIA for the synthetic and MCSL parts of traffic pattern B.

we do not address congestion directly but we avoid HoL-blocking. Thanks to this, we achieve good results in keeping latency and throughput of regular traffic, obtaining up to three times better latency when compared with the same scenario without BAHIA. As future work we plan to deal with internal congestion within the network and solve the problem in the same way, by eliminating at runtime the HoL-blocking introduced by congested traffic.

Acknowledgment

This work has been supported by the project NaNoC (grant agreement no. 248972) which is funded by the European Commission within the Research Programme FP7, by the Spanish MICINN, Plan E funds, under Grant TIN2009-14475-C04-01, by the Junta de Comunidades de Castilla-La Mancha under project POII10-0289-3724, and by the Spanish Ministerio de Economía y Competitividad (MINECO) under Grant TIN2012-38341-C04-01, and by Programa de Apoyo a la Investigacin y Desarrollo (PAID-05-12) of the Universitat Politécnica de València under Grant SP2012.

Chapter 5

ICARO: Congestion Isolation in Networks-On-Chip

- **Authors:** José Vicente Escamilla (Universitat Politècnica de València), José Flich (Universitat Politècnica de València) and Pedro J. García (Universidad de Castilla-La Mancha)
- **Type:** Conference
- **Conference:** 8th IEEE International Symposium on Networks-on-Chip (NoCS)
- **Location:** Ferrara, Italy
- **Year:** 2014
- **DOI:** 10.1109/NOCS.2014.7008775
- **URL:** http://ieeexplore.ieee.org/document/7008775/
- **Citation:** [77]

5.1 Abstract

The growing demand of computing power and the emerging trend towards heterogeneity lead to integrate more and more cores and specialized modules into a single chip. As the number of cores per chip increases, the network interconnecting them must satisfy the growing communication needs. However, several factors as aggressive traffic patterns, power-saving and fault-tolerance mechanisms may lead to oversubscribed resources in the network, thereby generating congestion and so degrading the overall network performance. In this paper we propose ICARO, a mechanism to dynamically isolate the traffic flows contributing to congestion making use of dedicated virtual networks. In this way, ICARO prevents the head-of-line blocking effect derived from congestion, thereby improving overall network performance. We analyze thoroughly our proposal, especially from the robustness point of view, showing that it effectively manages to identify and isolate congested flows, improving network performance up to 82% with respect to previous proposals.

5.2 Introduction and Motivation

Nowadays, High-Performance Computing (HPC) and multimedia-oriented applications and services demand increasing computing power to the systems supporting them. Moreover, achieving this performance with minimum power consumption has become almost mandatory due to cost and power constraints. In order to satisfy both requirements, manufacturers benefit from the advances in integration-scale technology, and include as many computing modules as possible into the same die. This leads to the design of chip multiprocessors (CMPs) or multiprocessor systems-on-chip (MPSoCs). To date, it is common to find devices with tens or hundreds of modules [60][7][78]. Although this approach offers flexibility, it does not offer high performance for specific usual computing tasks. As a consequence, heterogeneous designs are being revisited as they may include both multi-purpose and specific-purpose modules such as cache pre-fetchers [79], accelerators [80], etc.

Regardless the specific design, these platforms require an interconnection network to support communication between all the processing nodes. In general, this network must provide high-bandwidth and low-latency to avoid slowing down the processing nodes while waiting for remote data. In that sense, Networks-on-chip (NoCs)[5] are well suited to systems with a high number of processing nodes. NoC design presents interesting challenges due to the tight constraints found in this environment. Among these challenges, an still open issue is how to efficiently deal with *congestion situations*, i.e.

[0]This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2012-38341-C04-01 and by Ayudas para Primeros Proyectos de Investigación from Universitat Politècnica de València under grant ref. 2370.

scenarios where any number of the network internal paths are clogged, mainly due to oversubscribed ports (hotspots). Basically, an oversubscribed port is a port requested concurrently by several traffic flows, so only one is granted access at a given moment while the rest will be temporarily blocked. This competition to access the ports is usually known as *contention*, congestion being actually the result of persistent contention, whose effects (i.e. blocked flows) are propagated throughout the network due to the backpressure of flow control. Indeed, congestion may lead to a severe degradation of network performance if no countermeasures are taken.

Such congestion situations are likely to appear in NoCs due to several causes. For instance, the nodes in heterogeneous systems may generate bursty traffic, thereby increasing the probability of hotspot appearance. In addition, in such systems, due to the different nature of each node, the traffic generated is inherently unbalanced, so being prone to create hotspots that degrade network performance. Nevertheless, bursty traffic is not the only case of traffic patterns creating hotspots; indeed, any application involving intense communication towards more-preferred destinations may create them. Besides, some normal operations prone to be concurrently demanded, such as accessing the main memory controllers, may "naturally" create hotspots. In addition, fault-tolerance mechanisms may re-route traffic flows to recover from network failures [81], thereby probably causing an unbalanced traffic distribution, and so leading to oversubscribed links and ports.

Moreover, as mentioned above, power consumption has become critical in HPC systems due to their large number of nodes and their high clock frequency, which lead to high costs, even to dissipate the heat generated by the machines. In the case of portable computing devices (like multimedia devices, smartphones, etc), power consumption is the main pending issue due to the insufficient capacity of their batteries. Indeed, in such devices, one of the main power-hungry parts is the MPSoC [82], becoming mandatory to improve its power consumption. To achieve this, CMP and MPSoC manufacturers have developed and implemented mechanisms to reduce the energy necessary to keep working these devices while reducing their performance as less as possible. One of the most extended mechanisms to carry this out is Dynamic Voltage and Frequency Scaling (DVFS) [3][83]. However, switching from low to high voltage/frequency values may not be fast enough to satisfy a sudden increase in network performance demand, thereby causing transitory network congestion as the network offered bandwidth is temporarily insufficient.

Overall, any of the aforementioned causes of congestion may end up degrading network performance, especially due to the Head-of-Line (HoL) blocking effect associated to congestion situations. Specifically, HoL-blocking arises when flits belonging to messages requesting oversubscribed ports precede in FIFO queues to flits belonging to messages requesting other ports, which may be free. While the head-of-line flits do not win their requested ports, such flits force the ones stored behind to stay blocked in the queue, even

if the latter request available ports. It is worth describing this effect because the mechanism we propose in this paper is based on the premise that congestion is not a problem by itself, the actual problem being the HoL-blocking caused by congestion. Indeed, if HoL-blocking is completely prevented, network performance is not affected even though the traffic flows contributing to congestion (hereafter referred to as congested flows) are not removed, as some techniques proposed for off-chip networks have demonstrated (see Section 5.3). By contrast, this principle has not been yet efficiently applied in the NoCs context. To fill this gap, in this paper we propose ICARO (Internal-Congestion-Aware Hol-blocking RemOval), a novel mechanism to prevent HoL-blocking in NoCs, which uses special Virtual Networks (VNs) to dynamically isolate congested flows, separating them from non-congested ones. In this way, the HoL-blocking derived from congestion is prevented by using a reduced set of network resources (VNs), thereby cost-efficiently improving network performance.

The rest of the paper is organized as follows. In Section 5.3, we offer an overview of existing approaches to deal with HoL-blocking. Next, ICARO is described, dividing the mechanism into three clearly differentiated stages. In Section 5.5 a deep behavioral analysis of ICARO is performed by testing its robustness and scalability, and by comparing its performance improvement against other similar proposals. Next, we analyze the area and power overhead of ICARO through HDL implementation results. Finally, in Section 5.7, some conclusions are drawn and the future work on ICARO is indicated.

5.3 Related Work

Due to the negative impact of congestion, and also to the growing popularity of NoC-based systems, the number of proposals for congestion management in NoCs has quickly increased during the last years. Although some congestion-management mechanisms have been proposed for bufferless NoCs, such as the one presented in [16], hereafter we focus on the solutions oriented to buffered NoCs, as ICARO has been designed for this type of NoC architecture.

Many of the solutions for buffered NoCs are based on collecting congestion information from neighboring nodes through the routing process and monitoring buffer occupancy, in order to offer an alternative path to route around congested areas (i.e. hotspots). Among them, a mechanism called RCA is proposed in [20] for congestion avoidance in NoCs with adaptive routing. RCA uses a composition of multiple global metrics collected from the whole network to select at each router the output port which messages are forwarded through, so that hotspots are avoided. Specifically, these metrics are: the count of free Virtual Channels (VCs), the count of free buffers and the crossbar demand. In order to collect the metrics from the whole network, such metrics are aggregated (piggybacked) from a router to the next one and so on. In a heavy-congestion situation this mechanism

may collapse since the information used to avoid the congested areas is aggregated in the same messages that are congested, so a vicious cycle may be created. However, adapting the routes to avoid hotspots may result in moving the location of such hotspots from one place to another, so the problem would remain unsolved. Moreover, avoiding hotspots may be impossible if all the congested flows have the same target (e.g. the memory controller).

Another solution based on adaptive routing policy is PARS, proposed in [22], which uses a dedicated subnetwork for sending congestion metrics based on the buffer state at certain routers. Like RCA, PARS uses such metrics to select proper paths in order to avoid hotspots. Although in this case the information is sent through the dedicated subnetwork, the problems regarding unavoidable hotspots or "hotspot reallocation" may still appear. Similarly, in [23] authors propose a token-based flow-control mechanism which uses dedicated wires to send routers status information (token) which is used to take routing decisions and bypass routers pipeline. However, this proposal is focused on reducing network latency by skipping routers pipelining, but not by facing congestion harmful effects. In [21] authors propose to collect congestion information from the whole network and to take routing decisions based on network status. However, in this proposal the congestion information is collected piggybacking the links status into the packets header.

Following a different approach, a predictive-based flow control mechanism is proposed in [19]. Authors propose an end-to-end flow control mechanism based on prediction-models to control the injection rate at the source node. Predictions are computed in every switch using its state and its neighbors state. In order to exchange the necessary data for computing the prediction, routers implement additional wires interconnecting them. This solution actually corresponds to one of the "classical" approaches to congestion management, usually known as *injection throttling*, which, like any control strategy based on closed-loop theory, may present performance oscillations and become inefficient if the source nodes react too late. Similarly, in [24] authors propose HPRA, a hotspot-formation prediction mechanism, that makes use of an Artificial Neural Network-based (ANN) hardware that gathers buffer utilization data to predict the formation of hotspots. Then, HPRA classifies the traffic into two classes: hotspot-destined traffic (HSD) and non-hotspot-destined traffic (nonHSD). HSD traffic is throttled at source while the nonHSD traffic is routed avoiding paths containing hotspots routers. This proposal may suffer from the same problems as *injection throttling*.

An alternative approach is to specifically deal with the HoL-blocking derived from congestion, mainly by mapping different traffic flows to different queues in the buffers, so that the interaction between flows is minimized. A solution that follows this approach has been proposed in [25]. Actually, authors propose two policies to map traffic flows to VCs: FVADA and AVADA. Both proposals establish a correspondence between the output port requested on the router *x+1* and the output VC assigned in the router x

(note this requires *lookahead routing*). The main difference between both policies is that FVADA establishes a direct and constant correspondence between the requested output port and the assigned VC, while AVADA starts establishing a direct correspondence but later this correspondence can be dynamically adapted, based on the output port load, making use of a correspondence table (a CAM-based table). Note that, while FVADA is simpler to implement, it requires exactly as many VCs as the $router\_radix - 1$ value, thereby the number of required VCs depending on the router radix. Moreover, both policies require routers implementing lookahead routing and a credit-based flow-control in order to quantify the output port load an adapt their behavior when the load in a given VC is too high. Note that neither FVADA nor AVADA are actually aware of which traffic flows are contributing to a hotspot, as they only consider one hop (i.e. the next requested output port) in the path of the messages, while hotspots may be located further away. Thus, congested flows may still share queues with non-congested ones, thereby still causing HoL-blocking in some degree. A different approach to deal with HoL-blocking is proposed in [84], based on an Unified Buffer Structure in which the number of buffers per port and their depth are allocated dynamically depending on the traffic load, dispensing fewer but deeper VCs under low traffic loads and more but shallower VCs under heavy traffic.

Thus, we believe that an efficient HoL-blocking-avoidance mechanism must explicitly identify congested flows in order to isolate them completely and dynamically. This is the approach followed by the Regional Explicit Congestion Management (RECN) mechanism, proposed for off-chip networks [15]. Among the plethora of proposals for congestion management in off-chip networks, RECN can be considered as one of the most efficient as it completely prevents HoL-blocking while requiring a reduced set of queues. However, adapting the RECN basics to NoCs requires a very different way of implementing it, due to the tight limitations in area and power in this context. In that sense, in [59] a solution is presented to isolate bursty traffic, but not congested flows. In this paper we finally propose a solution, ICARO, that adapts the RECN strategy to the NoCs context, as explained in the next section.

5.4 ICARO Description

5.4.1 ICARO Principles

As mentioned in the previous sections, the purpose of ICARO is not removing congestion but preventing the HoL-blocking caused by congestion. Indeed, ICARO manages to solve this problem by identifying congested flows, then isolating them into special Virtual Networks (VNs) while keeping the non-congested data flows in different regular VNs. By doing this, ICARO separates congested flows from non-congested ones, thus preventing HoL-blocking and so increasing network performance. Note that ICARO needs at least

two VNs: one *regular-VN* (for non-congested traffic) and one *slow-VN* (for congested traffic). Nevertheless, ICARO may be configured to work with several regular-VNs and also with several slow-VNs. Note that ICARO is a reactive mechanism in the sense that all the system works normally in absence of congestion, keeping the system performance in the same values as the baseline (i.e. the same scenario without ICARO). However, when congestion is detected ICARO reacts to keep network performance by preventing the congestion harmful effects.

ICARO functionality can be divided into three stages: first, congested points in the network are detected at routers; then routers notify the sources of this detection; finally, the sources map the traffic flows either to a slow-VN or to a regular-VN depending on whether or not the injected flow will traverse congested points. These three stages are thoroughly described in the next subsections.

5.4.2 Congestion Detection

ICARO is based on detecting *congested points*, defined as output ports persistently oversubscribed. According to the definition of contention given before, ICARO considers that exists contention for an output port if two or more flows request that output port from different input ports. In order to detect whether this contention is persistent, an additional metric is used. This metric consists in counting the number of messages requesting the contended output port. This count is computed per VN at input ports, increasing its value when a new message requesting the output port arrives to the input queue, and decrementing it when the whole message leaves the queue. Every time this count is modified, it is compared with a threshold (SAT_THR) whose value is a configurable parameter of ICARO. Depending on the value of SAT_THR, the congestion-trigger sensitivity of ICARO is lower or higher. As each input port may contain several VNs, an input port is considered as exceeding SAT_THR for an output port if anyone of its VNs does it. Therefore, an output port is considered as a congested point when, in two or more input ports, there are VNs exceeding SAT_THR for that output port.

ICARO must also detect the end of congestion. For this, an hysteresis technique is used. In a few words, once an output port is detected as congested, ICARO detects the end of congestion when the number of messages requesting that output port (in all the VNs) falls below the UNSAT_THR threshold, being UNSAT_THR $<$ SAT_THR. The pseudo-code in Algorithm 3 describes the whole mechanism.

5.4.3 Congestion Notification

Once a congested point is detected (or when a previously congested point is no longer congested), sources must be notified in order to isolate (or stop isolating) congested flows

```
for each output_port do
    for each input_port do
        port_saturated = FALSE;
        for each vn do
            if isVNsaturated(input_port, vn)==TRUE then
                if getNumRequests(vn, output_port) < UNSAT_THR then
                    port_saturated = FALSE;
                    markVNasUNSaturated(input_port, vn);
                else
                    port_saturated = TRUE;
                    break;
                end
            else
                if getNumRequests(vn, output_port) > SAT_THR then
                    port_saturated = TRUE;
                    markVNasSaturated(input_port, vn);
                    break;
                else
                    port_saturated = FALSE;
                end
            end
        end
        if port_saturated==TRUE then
            num_ports_saturated++;
        end
    end
    if num_ports_saturated >= 2 then
        markAsCongested(output_port);
    else
        markAsNoCongested(output_port);
    end
end
```

Algorithm 3: Congestion/no-congestion detection algorithm.

into slow-VNs. To deliver this notification ICARO employs a simple dedicated network called CNN (Congestion Notification Network) to send data about the status of the ports to all NIs in the network. The CNN consists in a P-bits-width ring-network to which all NIs and routers are connected, being $P = log_2(NumNodes) + Router\_Radix + 1$. Obviously, in the CNN routers act always as injectors and NIs as receivers so there is no media-access conflicts between NIs, but there may be conflicts between routers. To solve this, this ring is segmented by registers, so that each router has an associated register which separates the signals coming from the previous router from the signals being injected from the current router to the next one. The signals at each register are propagated to the next one at each clock cycle. An schematic definition of two consecutive routers is shown in Figure 5.1.

When a router needs to inject data into the network, it waits until its associated register gets free. In practice, the router just keeps the port-status signal at the input of a multiplexer, which selects the register input signal depending on the busy bit sent by the previous register. When the current router register gets free, the multiplexer injects the signal to the register and such signal is propagated to the next register at the next cycle, and read by the next NI at the same time. When the port status data generated from router x returns to the router x (the data has completed the loop along the ring), such data is dropped and the register x is freed. A complete CNN is shown in Figure 5.2.

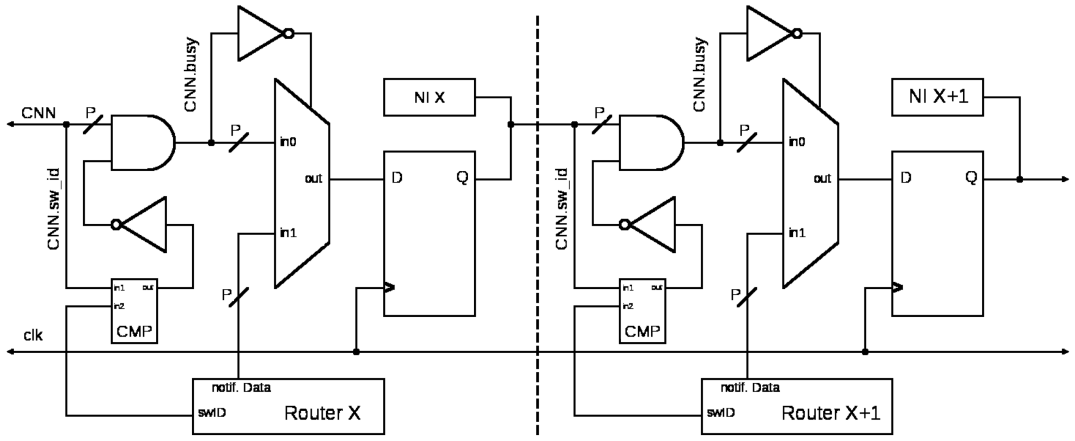


FIGURE 5.1: CNN registers example.

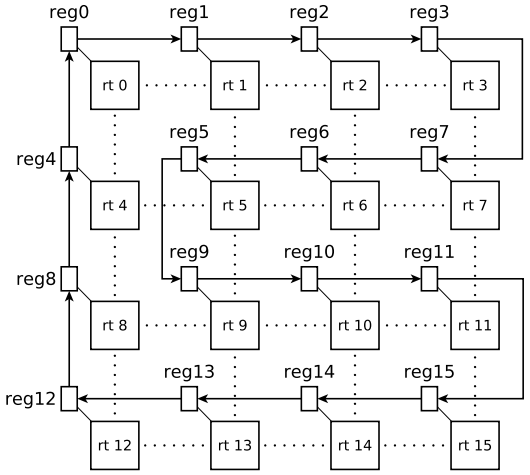


FIGURE 5.2: Complete congestion notification network (CNN).

The congested-point data sent through the network consists of the router ID coded in binary, a bitmap corresponding to all ports in the router (a bit set to 1 means that the port corresponding to the bit position is congested, otherwise the port is not congested), and an additional bit set to 1 to indicate a valid signal (busy bit). All data is transmitted in parallel, so the CNN must be P-bits-wide.

As some congestion notifications may be dropped at Network Interfaces (NIs) (explained later) or simply lost due to transient failures, the status of the ports is transmitted regularly (re-sync mechanism) to keep the congested-points data coherent at NIs. The frequency at which such data is transmitted is a configurable parameter of ICARO.

Note that this mechanism may not scale for very large systems, as notifications may take too much time to reach all nodes in the network, this delay spoiling the performance improvement achieved by ICARO. Thus, for large systems, instead of using an

unidirectional ring to deliver notifications, an hierarchical rings arrangement could be used. However, the evaluation of this option are left for further work.

5.4.4 Congestion Isolation

Congestion isolation is performed at NIs. As commented in the previous sections, ICARO makes use of at least two VNs: one slow-VN and one regular-VN. All flows are always mapped first to regular-VNs, but a module called *post-processor* is in charge of checking the head of all regular-VNs to find messages that should be re-mapped to slow-VNs. The post-processor checks all queues each cycle: If it finds a head of message, the destination is analyzed in order to check whether or not this message will traverse a *congested point*. If so, the message is re-mapped to a slow-VN. In case of having more than one slow-VN available, the re-mapping module follows a *modulo-mapping* [49] strategy. Since we make use of VNs instead of VCs, messages injected to the network through a given VN are never moved to other VN, as this is the key of isolation mechanism. Messages are provided in their header with a *VN id* prior injection. Such *VN id* is read by routers along the path in order to know which VN the message must be mapped to. The NI arbiter can be a typical arbiter (such as a *round-robin* arbiter). Note that the re-mapping mechanism can be executed in parallel with the injection of a message from other VNs except the one from which a message is being re-mapped. Note that ICARO is intended to be used with deadlock-free deterministic routing algorithms (e.g. XY), so no deadlocks can arise.

5.4.4.1 Congested-points Cache

NIs must implement a mechanism to manage and store the congested-points data that must be available for the post-processor. This mechanism mainly consists in a cache memory and some additional logic. Figure 5.3 shows a diagram explaining the notification storing process. When a notification arrives to the NI, first this notification is *deserialized*, then traverses some filters (contained in the *Notification-processing* module in Figure 5.3), and is finally stored in the cache. This cache may be implemented as flip-flop registers and it is arranged in several rows and two columns, each row corresponding to a notification while each column corresponds to the data fields contained in the notification (router ID and port). As explained previously, notifications arrive through the CNN as a router identifier coded in binary and a bitmap describing the status of each port. However, not all the port-status data is relevant to the NI since not all ports of a given router are reachable from the receiver NI. Because of this, and also to speed up the cache queries, each notification received through the CNN is split internally into as many notifications as the router radix value (for being able to discard individual port-status notifications). Once the router notification is split into port

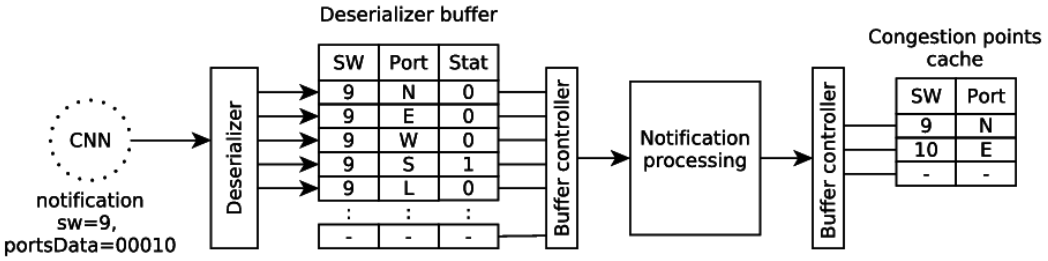


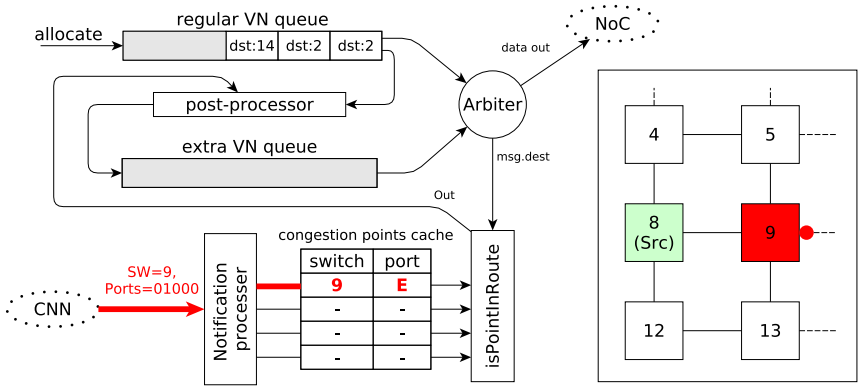
FIGURE 5.3: Notification management.

notifications made of *router, port_status* paired values, each notification is stored in a temporary buffer (*deserializer buffer*). This buffer receives a notification containing the status of the whole ports of the router at once, allowing the next functional module to read and process each port notification one by one at each cycle. This allows to receive and process properly one notification (containing all ports status) at each clock cycle through the CNN (in extreme cases) during a lapse of time, depending on this temporary buffer size. In case of *deserializer-buffer* overflow, notifications are dropped relying on the *re-sync mechanism* that allows to receive and process them later safely.

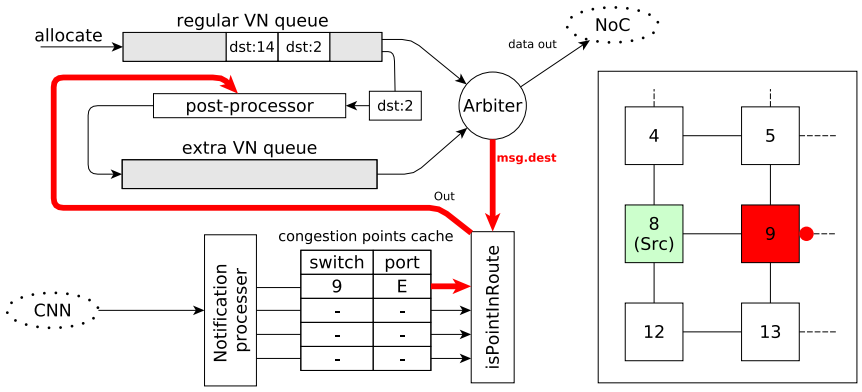
Once the notifications are stored in the *deserializer buffer*, each notification traverses a filter which discards unreachable points from this NI, thereby optimizing cache utilization. Congestion notifications which pass the filter are stored in the cache. End-of-congestion notifications which pass the filter trigger a matching-mechanism that removes from the cache congestion notifications which match the same router and port.

In Figure 5.4 an example of an ICARO NI working in a 4x4 2D mesh is shown. The example shows the behavior of the NI 8 in the network. As can be seen, in Figure 5.4a the NI receives a notification from router 9. This notification contains the router ID and the port status bitmap which informs that the East port of such router is congested while the other ports are in normal state. As shown, the notification is processed and stored in the cache. Next, in Figure 5.4b the message destined to the NI 2 tries to be injected into the network but, as the message will traverse the East port of router 9 (in order to arrive to the NI 2), the message is not injected, instead being re-mapped by the *post-processor* to the *slow-VN*, which the message will be later injected through. Finally, in Figure 5.4c the NI receives a new notification from router 9 informing that none of its ports is congested, so the stored notification is removed, therefore no more messages will be re-mapped to the slow-VN.

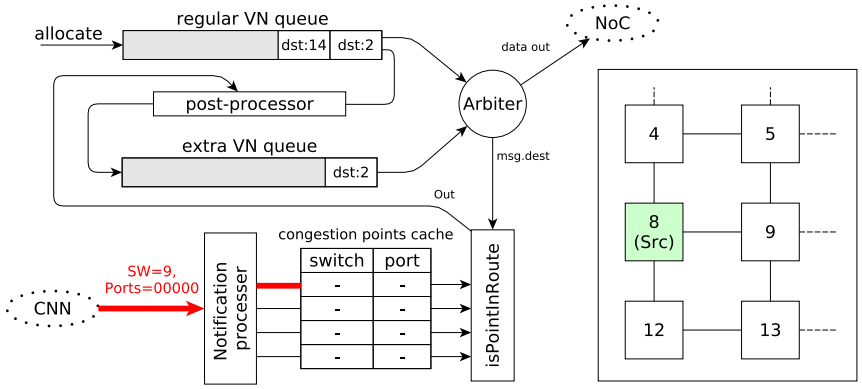
Note that, although ICARO makes no use of the slow-VNs in absence of congestion, this does not necessarily lead to lower performance in terms of latency, as the number of VNs only affects latency when network capacity reaches its limit, and ICARO would start using all VNs in this case.



(A) ICARO receives a congestion notification.



(B) A message is reallocated to the slow-VN.



(C) ICARO receives an end-of-congestion notification.

FIGURE 5.4: ICARO NI module mechanism description.

5.4.4.2 Optimizations

To optimizing even more the cache utilization, a congested-points *merge* mechanism can be used. Congestion tends to spread over the network starting from a root point. If congestion spreads towards a given NI, routers contained in the path to the root may notify for congestion sequentially. Due to this, the NI cache may be populated of multiple congested points contained in the same route so all of them but one are

redundant since with only the one closer to the NI, the congested route would be covered completely. Thus, an optimization of ICARO consists in a merge mechanism in order to allow the redundant notifications to be discarded in a second filter before to storing new congested points. Let us suppose that we have a notification already stored in the cache (notification A). When a new notification arrives (notification B) three circumstances may occur: the new congested point is covered by another, already stored congested point (Figure 5.5a), the new congested point covers one or more already stored congested points (Figure 5.5b), the new congested point belongs to a new route (Figure 5.5c).

In the first case, the new congested point is useless because all messages crossing B will cross A so the new notification is redundant, therefore can be discarded safely. In the second case, A is contained in B, so A becomes useless if we store B, therefore, A is replaced by B. Also, in such case, more rows may be affected by B. B may cover several already stored congested points, so such congested points can be safely merged, discarding them and storing B instead.

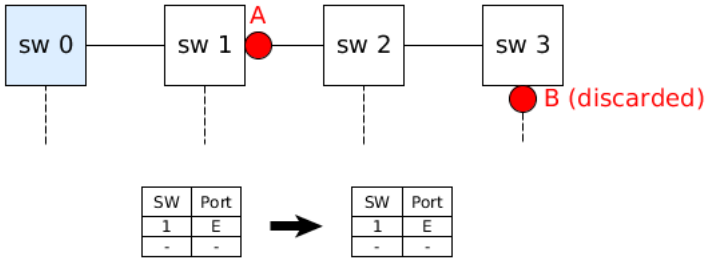
However, despite this optimization achieves good results in minimizing the cache utilization, in some scenarios may be counter-productive in performance terms. When using the merge mechanism, the stored congested points tend to get closer to the NI, thereby isolating too much traffic into the slow-VN. Also, since congested points belonging to branches of the congestion tree are usually more volatile than the congestion root, such congested points disappear quickly, thereby removing the congested points at NIs and so becoming the mechanism unstable in some cases. Therefore, this optimization should be used only when the cache size must be critically small due to the lack of silicon area available.

5.5 Performance Evaluation

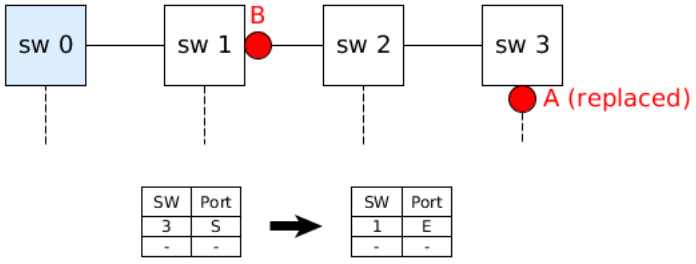
For evaluating the performance we use a network-on-chip cycle-accurate simulator developed in our group. The ICARO results are compared with FVADA and AVADA [25]. First, the scenarios used for the simulations are described. Next, ICARO is evaluated in different scenarios varying critical parameters that may affect its performance. Then, a performance analysis is carried out comparing ICARO with FVADA and AVADA.

5.5.1 Simulation Environment

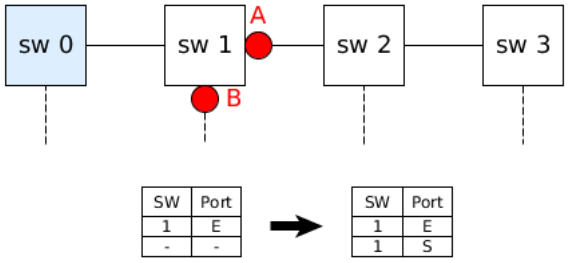
For our simulations we use a 4-stage pipelined router: IB (data storing into the buffer), RT (routing computation), VA/SA (VC allocation/switch allocation, both running in parallel), X (crossbar). Our router uses wormhole switching with flit-level crossbar switching and implements credit-based flow control. The size of the router queues is



(A) A already covers B, so B is discarded.



(B) B covers A, so A is replaced by B.



(C) A and B belong to separated routes.

FIGURE 5.5: Merge opportunities.

16. The amount of queues varies depending on the strategy used, e.g. if the amount of queues is 4, the total slots for a given input port will be $4 * 16 = 64 flits$. A regular 8x8 2D mesh network is used (with XY routing), so router radix=5. It is noteworthy that, both FVADA and AVADA make use of virtual channels (VCs) instead of VNs as ICARO does. So, for the baseline scenario we decided to make use of VCs as well. Messages are 5-flits long with a flit size of 128 bits. Regarding traffic patterns, two types of synthetic traffic patterns are used. On one hand, typical synthetic traffic patterns are used like uniform, tornado, bit-reversal, etc. On the other hand, as ICARO is a proposal intended to deal with irregular, bursty, hotspot-prone traffic patterns, we drew up a combined traffic pattern. This traffic pattern is composed of a light uniform background traffic and a hotspot component. Hotspots consist in several nodes (the amount depends on the network size) receiving each one high data rates from 4 nodes (4-to-1 hotspots). Hotspots are active only from cycle 10k to 20k. In this way, we have a background traffic which generates no congestion, and another component of aggressive traffic which

causes congestion, causing HoL-blocking to the background traffic.

5.5.2 Robustness Analysis

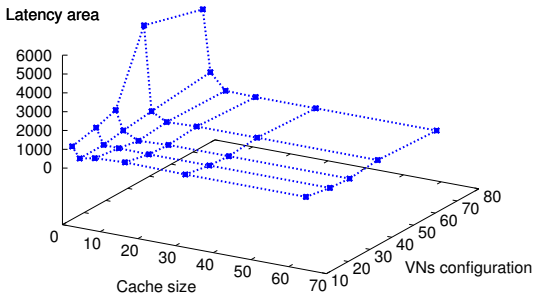
In previous sections we stated that ICARO needs at least 2 VNs, one for regular traffic and one for congested traffic. However, the amount of VNs can be increased as much as we need, arranging the VNs in several configurations: 1+1VNs, 2+2VNs, 4+1VNs, 4+4VNs, etc. Also, for ICARO, the cache size is a critical parameter depending on the traffic pattern and network size. So, for the purpose of evaluating the impact of such variables, an analysis is performed. For this analysis the combined hotspot traffic patterns have been used with a background traffic of 0.3flits/cycle/node.

The evaluation is performed with different VNs configurations. In order to graph the network latency for the different configurations, each VNs arrangement has a number assigned that identifies such configuration. Identifier XY is for a configuration with X regular-VNs and Y slow-VNs. We will play with configurations 11, 22, 31, 44 and 71. For the ICARO notifications we assume a propagation delay of 2 cycles for each hop. Regarding the baseline configuration we use exactly the same configuration as the ICARO scenario with the same number of VNs as in the ICARO case (considering both VN types: regular and slow VNs).

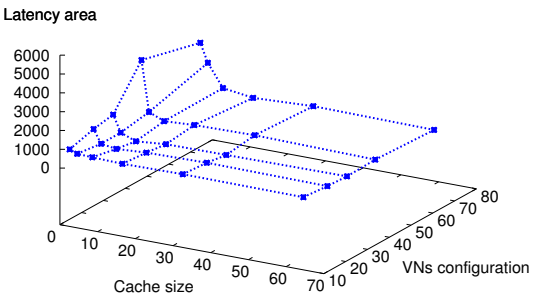
Regarding the SAT_THR and UNSAT_THR thresholds used in the congestion-detection mechanism (see Section 5.4), we performed simulations using different values for these thresholds, in order to evaluate their impact in ICARO and to obtain the optimal values. From the results obtained (not shown due to lack of space) we conclude that the thresholds values do not have a great impact on the ICARO behavior while they are confined in a reasonable range. Nevertheless, the best results are achieved for SAT_THR=4 and UNSAT_THR=2, so these values are assumed in the current analysis.

ICARO can work with the re-sync mechanism and/or the merge mechanism. Nevertheless, both mechanisms may cause counter-productive effects, so this analysis has been performed with three combinations of these mechanisms: *no re-sync/no merge, re-sync/no-merge* and *re-sync/merge*. The combination *no re-sync/merge* has not been considered as the purpose of the merge mechanism is to save cache slots in scenarios with a high number of notifications due to the re-sync mechanism.

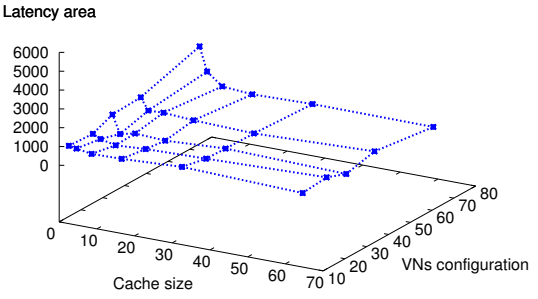
Regarding the *congested points cache*, the following sizes have been used: 2, 4, 8, 16, 32 and 360 (theoretical limit due to the maximum possible congested points that can be given in a 8x8 network). In the graphs, the 360 value has been replaced by 64 value for better viewing. For the *deserializer buffer* size we adopted the policy of using $2 * cache\_size$ entries.



(A) No re-sync, no merge.



(B) Re-sync, no merge.



(C) Re-sync and merge.

FIGURE 5.6: ICARO configuration analysis.

In Figure 5.6 the results are shown. The metric used for measuring the performance is the *latency area*, which consists of the sum of the latency overhead during the whole simulation. The latency overhead is measured as the difference of latency values between the case with congestion present and the case with only background traffic running.

As can be seen, in all cases, with 4 or more cache slots, there is no latency area increase as ICARO has room enough to store all relevant congested points, thus it is able to isolate all the congested traffic properly. However, as the number of cache slots available falls below 4, ICARO is not able to isolate congestion properly. However, as can be seen in Figure 5.6b, the re-sync mechanism helps to alleviate the shortage of cache entries. This is because congested points dropped due to the lack of room are re-notified periodically, so they have more opportunities to be stored. Besides, if we add the merge mechanism (Figure 5.6c), the latency falls even more as the congestion slots are better managed, so that there are more free slots to store congested points. However, the conclusion from this analysis is that with at least 4 cache entries HoL-blocking is completely removed regardless of VN configuration.

Parameter	Value
VNs config.	1+1, 3+1 and 7+1
SAT_THR	4
UNSAT_THR	2
Cache size	4
Deserializer buffer size	8
Re-sync	No
Merge	No

TABLE 5.1: ICARO configuration.

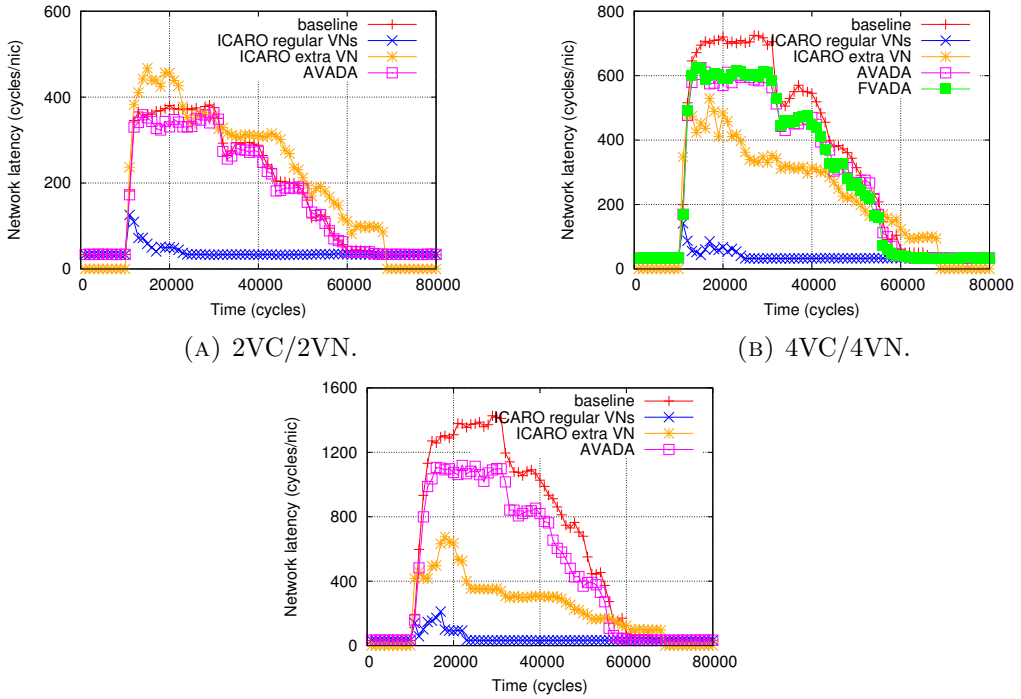
5.5.3 Overall Results

In this section the ICARO performance is compared against AVADA and FVADA. First, all techniques are simulated using common traffic patterns. As can be seen in Figure 5.8, for most of the common traffic patterns ICARO keeps the results in similar values to the baseline and the other techniques. In the case of ICARO there is a slight overhead close to saturation. This is due to the fact that it employs VNs instead of VCs. VCs gives more flexibility at the arbitration stage so is expected to perform better than using VNs. However, our proposal goal is not to improve the performance over static traffic patterns but with the combined hotspot one[1].

In Figure 5.7 we can see the latency results for the different mechanisms over combined hotspot traffic pattern. Note that all mechanisms but ICARO use VCs while ICARO uses VNs. In Table 5.1 the ICARO parameters configuration is shown. Let us recall that ICARO aim is to isolate harmful traffic into the slow-VN in order to avoid non-harmful traffic to be affected by the former. To better appreciate the ICARO behavior, latency results for our proposal are shown in two graphs: one for network latency average of all regular-VNs and another one for network latency of the slow-VN.

As can be seen, ICARO outperforms all other mechanisms achieving an improvement of up to 82% for the 8VN configuration. In the case of the 2VNs and 8VNs simulation FVADA is not shown because FVADA requires exactly *r-1* VCs (r=router radix). Notice that congestion injection lasts from 10k-cycle to 20k-cycle. The HoL-blocking effects in ICARO are minimized and removed after the congestion builds. However, for the other configurations, congestion remains beyond the 20k-cycle point. They recover performance point only beyond 60k-cycle.

[1]In the case of FVADA and AVADA, despite of reproducing exactly the same scenarios the authors used in their evaluations, we could not obtain the results exposed by them for common synthetic traffic patterns.



(A) 2VC/2VN.

(B) 4VC/4VN.

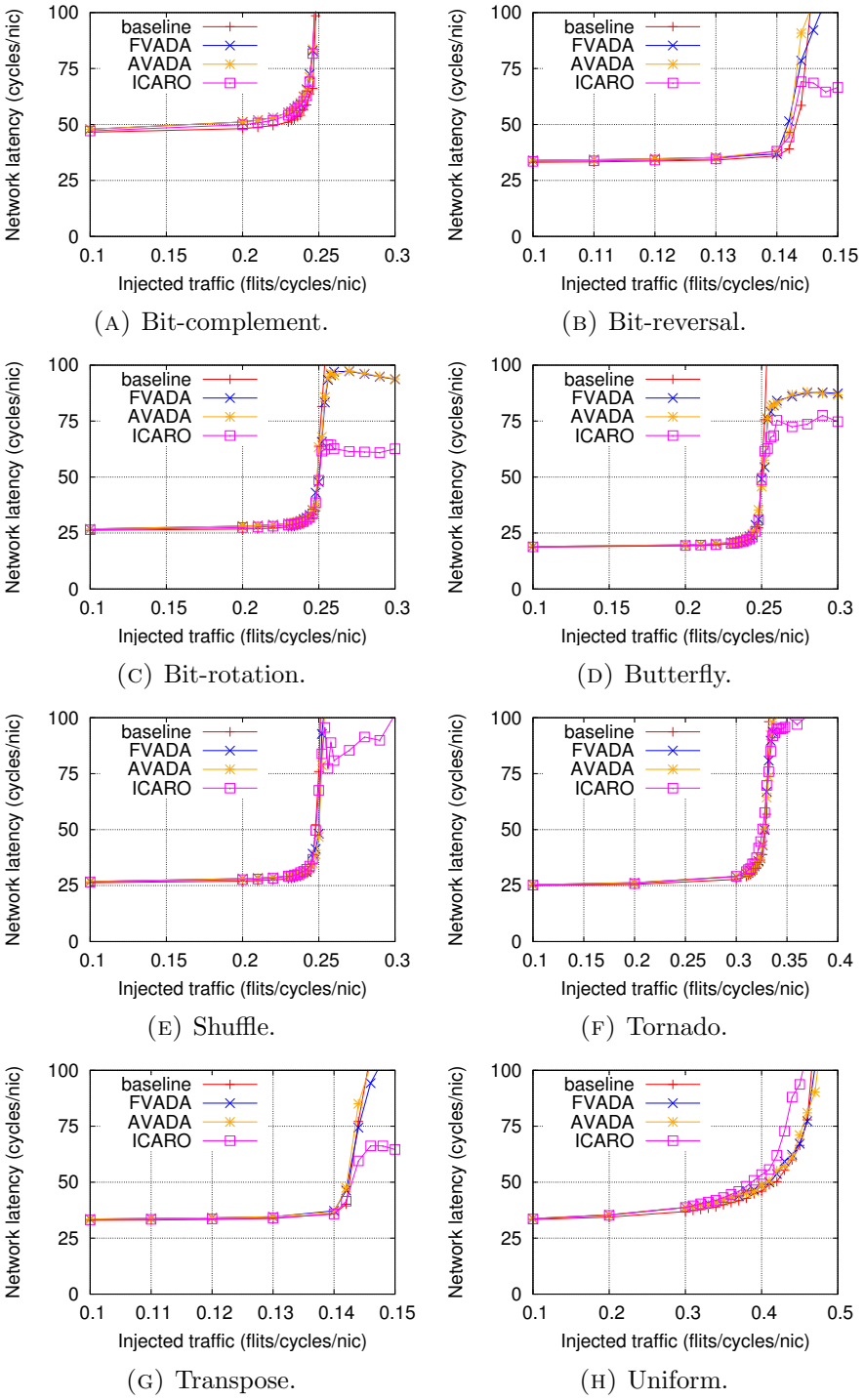
(C) 8VC/8VN.

FIGURE 5.7: Performance evaluation with hotspot traffic pattern.

5.6 Implementation Analysis

In this section, the area and power overhead of ICARO is analyzed. To perform this, the ICARO mechanism has been implemented in Verilog using a canonical NI and a wormhole router, both with support for 4 VNs. The router queues have a 4-flit size with a 128-bit flit size. For the NIs the queues have a size of 8 flits. Regarding the ICARO configuration, it has been implemented with support for merge and re-sync mechanisms with a cache size of 4 slots and a deserializer buffer size of 8 slots. To synthesize the Verilog designs, Design Vision tool from Synopsys with 45nm Nangate open cell library [51] (typical conditional) has been used. Then, we performed the place&route process with Encounter tool (from Cadence) to estimate accurately the area overhead. Figure 5.9 shows the results for the area and power overhead of a NI implementing ICARO compared with the baseline NI for different network sizes. In Figure 5.10, the area and power overhead results of our proposal for the router are shown.

ICARO needs additional hardware in order to implement the CNN. However, this hardware is not strictly located either at the router or the NI. This hardware consists of wires interconnecting nodes and the logic associated to these wires (shown in Figure 5.1). In order to fairly evaluate all the hardware overhead imposed by ICARO, the logic associated to the CNN is included in the router overhead. Wires are not taken into account as they do not actually impose area overhead. Indeed, such wires use metalization layers. However, in the design floor-plan, little empty gaps always exist between all tiles to



(A) Bit-complement. (B) Bit-reversal.

(C) Bit-rotation. (D) Butterfly.

(E) Shuffle. (F) Tornado.

(G) Transpose. (H) Uniform.

FIGURE 5.8: Typical synthetic traffic patterns.

physically isolate each tile from its neighbors. Provided that these gaps are big enough to physically place all links between tiles, the area spent by the whole CMP remains the same even including the CNN wires.

As can be seen in Figure 5.9, for all cases, the area overhead for the NI varies between 3.8% for a 16-node network, and 6% for a 1024-nodes network. For the power overhead

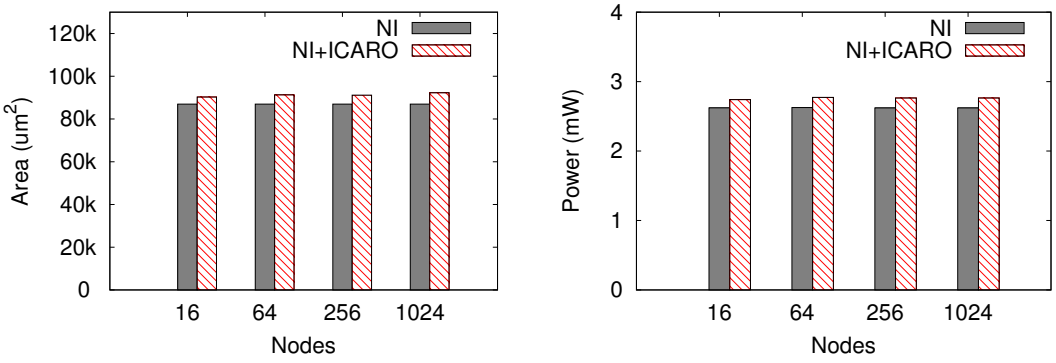


FIGURE 5.9: NI area and power overhead.

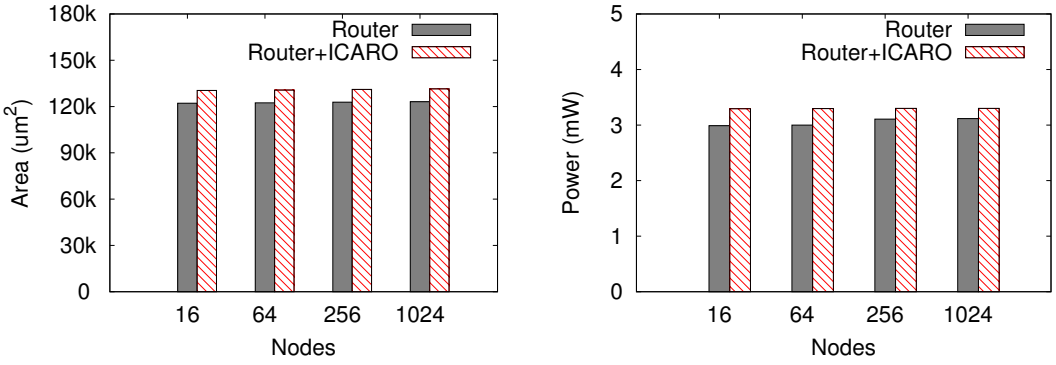


FIGURE 5.10: Router area and power overhead.

it varies from 4.5% to 5.4%. In Figure 5.10 the overhead results for the router are shown. For the area, ICARO has an overhead of 6.7% for all cases. In the case of power, values from 6% to 10% have been obtained.

As shown in such results, ICARO demonstrates an acceptable area and power overhead either for the NIs or the routers. In addition, taking into account the results for different networks sizes, seems clear that ICARO scales with the network size with no significant extra area or power overhead.

5.7 Conclusions and Future Work

In this paper a mechanism for avoiding HoL-blocking in NoCs has been presented. ICARO manages to identify harmful traffic and properly separates it from non-harmful one making use of VNs. This way, ICARO achieves improvements of up to 82% on the overall network latency with no significant area and power overhead. As future work we plan evaluate proposals for scaling the CNN for very large systems.

Chapter 6

Efficient DVFS Operation in NoCs through a Proper Congestion Management Strategy

- **Authors:** José Vicente Escamilla (Universitat Politècnica de València), José Flich (Universitat Politècnica de València) and Pedro J. García (Universidad de Castilla-La Mancha)
- **Type:** Conference
- **Conference:** Fourth International Workshop on On-chip memory hierarchies and interconnects: organization, management and implementation (OMHI)
- **Location:** Viena, Austria
- **Year:** 2015
- **DOI:** 10.1007/978-3-319-27308-2_28
- **URL:** https://link.springer.com/chapter/10.1007/978-3-319-27308-2_28
- **Citation:** [85]

6.1 Abstract

As technology advances, applications demand more and more computing power. However, achieving the required performance is not nowadays the single target, as reducing power consumption has become a key issue. In that sense, power-control mechanisms such as Dynamic Voltage and Frequency Scaling (DVFS) are introduced in order to dynamically adapt frequency and voltage to the actual computing-power demands of applications. However, these techniques may not be as efficient as expected due to delays caused by frequency-voltage changes. Furthermore, data flows generated at high rates may cross slow voltage-frequency islands, thereby leading to congestion inside the on-chip network. To alleviate this problem, we propose a combined DVFS and congestion management strategy. Specifically, the policies to adjust DFVS levels are tuned cooperatively with the congestion management strategy, leading to power-saving achievements of up to 26% and latency improvements for non-congested traffic of up to 43%.

6.2 Introduction

Nowadays, High-Performance Computing (HPC) and multimedia-oriented applications and services demand increasing computing power. In order to satisfy this demand, manufacturers take advantage of the advances in integration-scale technology to include as many computing resources as possible into the same die. This trend has led to advanced designs in manycore chips. Regardless the specific design, these platforms require an on-chip network (NoC) [86] to support communication among all the processing and/or storage nodes. The NoC must provide high bandwidth and low latency, otherwise processing nodes will slow down as they have to wait for long to receive necessary data. Hence, the design of the NoC presents unavoidable challenges.

Among these challenges, a still open issue is how to efficiently deal with congestion situations, i.e. scenarios where any number of network paths are clogged, mainly due to oversubscribed ports (hotspots). Indeed, congestion may lead to a severe degradation in network performance if no countermeasures are taken. Another challenge is reducing power consumption. Technology has reached power and thermal limits, thus limiting clock frequency and voltage. Moreover, in battery-powered devices, energy must be efficiently managed so as to maximize working time of the device. For these reasons, the current trend is to provide manycore chips with power-control mechanisms.

One of the most popular mechanisms is Dynamic Voltage and Frequency Scaling (DVFS)[3]. Basically, it consists in adapting the frequency and voltage based on the actual computing-power demand. Reducing the frequency and voltage leads to a significant reduction in power consumption, thus saving unneeded energy. However, DVFS must be carefully designed since reducing the working voltage and frequency may reduce also network

performance. Thus, finding out the optimal conditions to increase or decrease the working voltage and frequency is critical to achieve the best trade-off between power saving and network performance. Additionally, voltage and frequency changes are usually performed in steps or levels, and those changes cause severe power and delay penalties, thus, demanding for a proper policy.

A DVFS change causes inherently the system to halt for a small period of time (due to the electronic limits)[1]. Recent proposals shadow this effect by setting different DVFS regions, called Voltage and Frequency Islands (VFIs)[55][56]. In this way, frequency and voltage for a given island become independent as they are only driven by the metrics obtained from such island. From the efficiency point of view, VFI islands achieve an undeniable enhancement [55] as applications running concurrently may have different needs, some maximizing performance while others minimizing power. Nonetheless, VFIs still pose new challenges. Data flows may cross several VFIs working at different levels. Thus, the crossing from a high-frequency VFI to a low-frequency VFI will slow down the flow, potentially leading to a congestion situation appearing on VFI boundary. Moreover, congestion may be propagated throughout the high-frequency VFI network.

Summing up, DVFS-based systems need a proper policy to perform frequency-voltage transitions and, on the other hand, need to avoid congestion when VFIs are used. To address both issues, we propose adapting a congestion-control mechanism called ICARO[77] (Internal-Congestion-Aware Hol-blocking RemOval) to DVFS-based systems with VFIs. By doing this, performance is maintained despite of the DVFS-transition delays and congestion is alleviated despite of data flows crossing VFIs with different levels. ICARO congestion metrics will be used to implement the DVFS policy to perform voltage-frequency changes. We target different possibilities to plug a congestion control mechanism with a DVFS policy. Several solutions are presented which improve different key metrics, such as power consumption or message latency. Results show that we achieve improvements on network latency of 43% for non-congested traffic with a power overhead of approximately 8%. For the second solution a gain of 26% on power consumption, with an improvement on latency of 2% at the cost of losing throughput and, finally, the last proposal achieves an improvement on latency of 19% with a power saving gain of 20%. As networks-on-chip consume up to the 36% of the total chip power[8][29], the benefits of our proposal may improve substantially the overall power consumption.

The paper is organized as follows. Section 6.3 shows related work in DVFS and congestion-control mechanisms. Section 6.4 describes the ICARO-DVFS method. Section 6.4.6 shows analysis results for different DVFS scenarios combined with ICARO. Finally, Section 6.5 shows conclusions and future work.

6.3 Related Work

Related to DVFS, one key issue is the voltage-frequency regulator (VR) due to the high delays caused when changing the voltage-frequency level. DVFS regulators are designed either off-chip or on-chip. Off-chip regulators support high amounts of power, but they are slow. By contrast, on-chip regulators are very fast but expensive in terms of area and do not support much power. In [9], a hybrid scheme using both types of regulators is proposed. In systems using VFIs, the more VFIs are implemented, the more power efficiency is achieved, hence having one VFI per node would be the best case. Under this premise, in [83] authors propose a per-core VFI approach based on on-chip VRs. Despite authors state that area overhead would not be an obstacle to implement their proposal, in a newer study [9] they discard this approach due to the large area required to implement so many on-chip VRs, supporting their arguments on results obtained in [33]. An accurate DVFS model is described [1] for different real architectures, comparing with values from real systems. We use values of voltage/frequency levels (Table 6.1) and level change delays (Table 5) to model DVFS in our simulator.

Related to congestion, we find several solutions. Solutions for buffered NoCs are based on monitoring buffer occupancy and collecting congestion information from neighboring nodes. A congestion-free path is then used to avoid hotspot areas. In this way, RCA [20] uses multiple global metrics collected from the whole network to select at each router the output port which messages are forwarded through. However, a vicious cycle may be created since the information used to avoid the congested areas is included in the congested messages. Besides, adapting the routes to avoid hotspots may result in moving the location of such hotspots from one place to another. Moreover, avoiding hotspots may be impossible if all the flows have the same target (e.g. the memory controller).

Another solution based on adaptive routing is PARS [22], that uses a dedicated subnetwork to send congestion metrics about the buffer state at certain routers, then using these metrics to select paths that avoid hotspots. However, the problems regarding unavoidable hotspots or "hotspot reallocation" may still appear. Similarly, in [23] a token-based flow-control mechanism is proposed which uses dedicated wires to send router status information (token) used to make routing decisions and bypass router pipeline stages. However, this proposal is focused on reducing network latency but not by facing congestion harmful effects. In [21] authors collect congestion information from the whole network to make routing decisions. However, in this proposal the congestion information is collected by piggybacking the links status into the packets header. In the next section we describe ICARO [77] which attacks congestion in a more efficient manner.

6.4 ICARO-DVFS Implementation

6.4.1 Dynamic Voltage and Frequency Scaling

Our DVFS implementation changes voltage and frequency in levels, depending on the performance demand. Each level corresponds to a given pair of voltage-frequency fixed values (Table 6.1). Off-chip VRs are assumed.

DVFS level	$Voltage(V)$	$Freq(GHz)$	DVFS level	$Voltage(V)$	$Freq(GHz)$
Level 1	1.30	3.074	Level 4	1.15	2.281
Level 2	1.25	2.852	Level 5	1.10	1.932
Level 3	1.20	2.588	Level 6	1.05	1.540

TABLE 6.1: DVFS levels assumed in the ICARO-DVFS mechanism (obtained from [1])

DVFS levels are changed by monitoring the occupancy at router buffers. Every *poll_period* cycles across all the monitored routers sample and report their occupancy level. If any of the buffer exceeds a Q_h threshold the DVFS level is decremented (voltage and frequency are incremented). Accordingly, if all the buffers exhibit an occupancy below the Q_l threshold ($Q_h > Q_l$), the level is incremented (frequency and voltage are decremented).

6.4.2 Voltage and Frequency Islands

ICARO-DVFS supports VFIs, so voltage and frequency changes can be applied per VFI domain. Each VFI has its own VR which collects metrics from the routers in its domain and changes voltage-frequency accordingly. However, routers at the boundaries between VFIs must be carefully designed as data flows will cross different domains. To address this issue, we implement mixed-clock/mixed-voltage buffers[52][53] which enables to write and read at different frequencies.

Figure 6.1 shows the router implemented at a VFI boundary (see Section 6.4.6.1 for more information about routers architecture design). The *input buffer* stage is divided into two *sub-stages*, the write part belongs to the frequency domain of the upstream router while the read stage belongs to the current router frequency domain. In this way, the rest of pipeline stages (after IB) are able to work normally at its corresponding frequency. The same is performed for the flow control logic. As credit-based flow control is used, the credits buffer at the downstream router works at both frequencies, at the upstream router frequency for writing and at the current router frequency for reading.

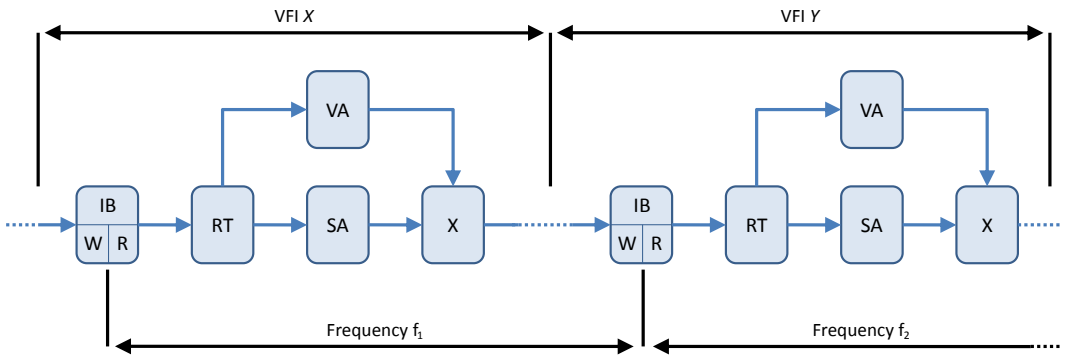


FIGURE 6.1: Two consecutive routers belonging to different VFIs (at the boundary delimiting such VFIs).

6.4.3 ICARO

Our proposal merges DVFS with ICARO [77], which tackles the congestion problem in a different way as the usual one. Specifically, ICARO focuses on reducing the impact of the Head-of-Line (HoL) blocking[14] derived from congestion situations. This harmful effect happens when congested flows share buffers with non-congested flows, then the former slowing down the latter and so degrading network performance. In order to deal with HoL-blocking, ICARO separates congested flows from non-congested ones by means of the different VNs implemented as disjoint virtual channels in the network. Note that congestion situations (i.e. congested flows) are not removed but their negative impact is reduced or even eliminated. Basically, ICARO detects congestion at routers, notifies congestion to the end nodes, and they react by steering packets through different virtual networks depending whether they cross congestion spots or not.

Congestion is detected at routers. A dedicated module at each router analyzes buffers of each port and, based on the buffer utilization, computes pending requests from each input port to each output port in order to detect contention (more than one request for a given output port). If contention is detected and caused by oversubscribed input ports (and lasts for a given threshold), then the output port is declared congested. Only if one output port changes its congestion state, then a notification is triggered from the router.

A congestion notification consists of the state of all output ports of the router (one bit per output port). Notifications are sent from routers to end nodes through a dedicated network (Congestion Notification Network or CNN) implemented as a k-width segmented

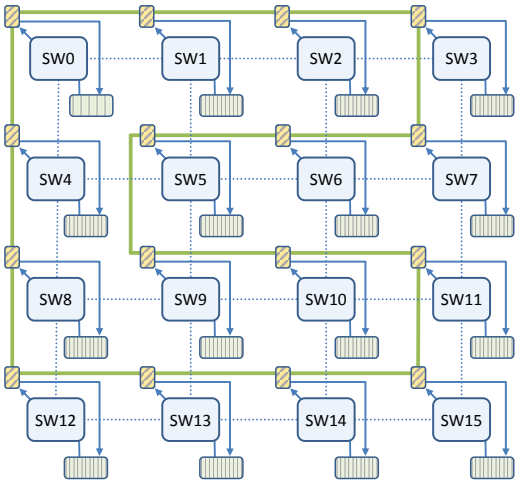


FIGURE 6.2: CNN network example. Links in green: CNN interconnecting all CNN registers.

ring where $k = log_2(nodes) + router_{radix} + 1$. Figure 6.2 shows a CNN implementation for a 4x4 mesh NoC. The CNN is made of N registers, each one owned by a router and connected to the next register through a k-width link. At each clock cycle, all the registers forward their data to the next register along the ring. If a router needs to inject a congestion notification, it checks its register state and injects the notification once the register becomes free. Notifications travel along the entire ring and are removed when they reach the register where they were injected from. Notifications are delivered to all end nodes and at each node the notification is processed and stored in a notification cache as *congested point* (pairs of values made up of {congested_router, congested_port}). Registers in this network are implemented as mixed-clock/mixed-voltage buffers as well to cope with different frequency domains.

Congestion isolation is performed at end nodes. To do so, ICARO uses two VNs: one congested-VN and one regular-VN. All flows are always mapped first to regular-VNs, but a *post-processor* module checks at each cycle the flit at the head of the regular-VN. If the flit is a header and its path travels through a *congested point*, it is considered a congested flow so, then the flit and the remaining ones of the same message are relocated in the congested-VN relocating automatically all flits from such queue until a tail flit is found). Messages injected through a given VN never change to a different VN, thus achieving the isolation property pursued by ICARO.

6.4.4 Merging ICARO with DVFS

Now we show how ICARO is coupled with DVFS. Basically, ICARO notifies congestion events through the CNN network. This network is extended to send DVFS notifications

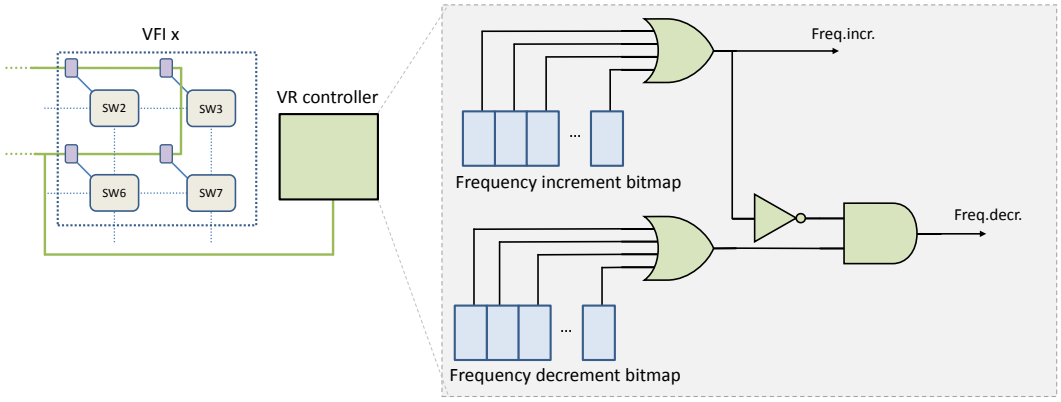


FIGURE 6.3: Voltage Regulator controller logic

as well[1]. This is easily achieved by changing the ICARO controller implemented in the router. In addition to detect congestion events, the new logic monitors all input ports queue occupancy and sends two new events through the CNN. Figure 6.4 shows the new notification format. Two bits are added just indicating the router requests for a level increment or decrement.

Buffer occupancy is analyzed in each router and compared against Q_h and Q_l thresholds. If any of the queues exceeds the Q_h threshold the VFI_{inc} bit is set. Once all queues occupancy are below Q_l the VFI_{dec} bit is set. Only when any of those two bits change a DVFS notification is sent through the CNN network.

At each VFI domain, the VFI module reads the notification commands from the CNN network, keeping record of all routers VFI_{inc} and VFI_{dec} bits from its domain. Two n-length bit vectors (being n the number of routers in the VFI domain) are implemented. VFI frequency/voltage is incremented when any of the routers request for such increment (even if a router is requesting for a decrement). Frequency decrement is performed when any router is requesting a decrement and none of the routers are requesting an increment. Figure 6.3 shows the logic, whereas Algorithm 4 shows the algorithm change DVFS levels.

6.4.5 Different ICARO-DVFS Alternatives

Besides the CNN extension, ICARO deals with different virtual networks (VNs) to decouple congested traffic from non-congested traffic. In the minimal implementation (2 VNs), just one VN is used to map congested traffic, the other one remains for non-congested traffic. As commented previously, to couple correctly ICARO and DVFS we

[1]In a typical DVFS implementation, a dedicated logic collects metrics from the network and deliver them to the logic that implements the VFI policy and drives its VR to carry this out. We take advantage of the CNN network to simplify this process.

```
if CNNreg.busy && idBelongsToVFI(CNNreg.routerID, thisVFI) then
    if CNNreg.FreqIncr then
        freqIncrVector[routerID] = 1;
        if |freqIncrVector then
            increaseFrequency(thisVFI);
    else if CNNreg.FreqDecr then
        freqDecrVector[routerID] = 1;
        if |freqDecrVector && ~ |freqIncrVector then
            decreaseFrequency(thisVFI);
    else
        Do nothing
    end
end
```

Algorithm 4: DVFS level change algorithm.

need to sense occupancy of input ports queues. However, as we have differentiated VNs we have different options.

As a first alternative, we can sense all input ports queues, including both non-congested traffic VN and congested traffic VN. In this case, DVFS will raise frequencies and voltages whenever traffic increases to levels where congestion appears in the VFI domain. This alternative is referenced to as *ICARO-2VN*.

Another alternative is to raise frequencies and voltages only whenever the non-congested VN congests as well. In this case, the DVFS strategy will raise only when severe congestion appears in the network. In other words, when congestion caused by hotspot traffic affects background traffic in such a way in which causes regular-VNs to exceed Q_h threshold. It is supposed that this alternative will lead to more power-saving results. This alternative will be referenced to as ICARO-1VN.

Finally, a different alternative is to sense all input port's queue, but differently from *ICARO-2VN*, the DVFS strategy will be bounded to a more conservative frequency level. In this case, the maximum frequency will be set to ∼2GHz (instead of ∼3GHz), corresponding to *Level 5* frequency on Table 6.1. The reasoning behind this strategy is the effect ICARO has on performance as will decouple congested traffic from non-congested one. Therefore, increasing frequency for performance reasons will become less critical. This alternative will be referenced to as *ICARO-2GHz*. In addition, this approach allows to simplify VRs by reducing the number of voltage-frequency levels provided. Notice that area consumed by VRs depends directly on the voltage-frequency levels provided.

The three strategies will be analyzed in Section 6.4.6. All of them will be compared against three different strategies. The two first strategies will not use DFVS at all and will set the network both to minimum and maximum frequencies. They will be referenced to as *minFreq* and *maxFreq*, respectively and will allow us to set the low and

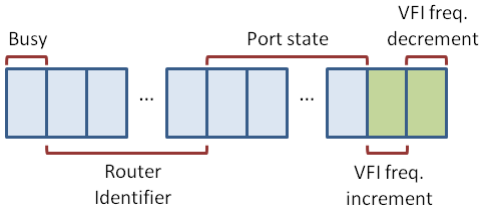


FIGURE 6.4: CNN signal format in DVFS-based platforms

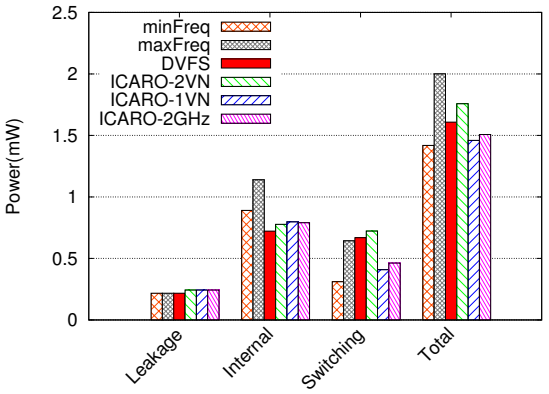


FIGURE 6.5: Final power consumption.

Topology	8x8 2D regular mesh
Routing policy	XY
Switching technique	Wormhole (flit-level)
Flow control	credits
Flit size	128 bits
Message size	5 flits
Switch queue size	16 flits

TABLE 6.2: Common simulation configuration.

up limits in terms of performance and power. The third strategy will be compounded of DVFS with the defined levels shown in Table 6.1 and sensing the VFI occupancy queues regardless of the congestion effects. This strategy will be referenced to as DVFS in the plots.

6.4.6 ICARO-DVFS Performance Analysis

6.4.6.1 Simulation Environment

We use gMemNoCsim, an in-house cycle-accurate event-driven NoC simulator. We model a 4-stage pipelined router: IB, RT, SA, X. At IB flits are stored into the buffer. In the case of routers at the VFI boundary, a *coupling IB stage* (see Section 6.4.2) is implemented with two substages: *IB_W* (writing data) and *IB_R* (read data). At RT, routing computation is performed, SA performs switch allocation, and at X stage the flit crosses the crossbar and leaves the router.

Regarding traffic patterns, as ICARO is a proposal intended to deal with irregular, bursty, hotspot-prone traffic patterns, we drew up a combined traffic pattern. This traffic pattern is composed of a light uniform background traffic (at a data rate of 0.01 flits/ns) and a hotspot component. Hotspots consist in several nodes receiving each one

high data rates (3 flits/ns) from 4 nodes (4-to-1 hotspots). Hotspots are active only from time 20ms to 40ms. In this way, we have a background traffic which generates no congestion, plus an aggressive traffic which causes congestion, causing HoL-blocking to the background traffic. This compound traffic pattern emulates environments where light data flows (i.e.: cores running applications with light data demand) share the NoC resources with heavy traffic generated nodes running high data demand applications or hardware accelerators which tend to generate heavy data bursts, causing congestion as well.

Simulations with DVFS are performed using real voltage, frequency and delay values shown in Table 6.1. For power consumption measures we use Orion 3.0[57].

6.4.6.2 Results

In this section results for different configurations are shown. First, two configurations without DVFS are considered: *minFreq* for the chip running at the minimum frequency (1.540GHz) and *maxFreq* for the chip running at the maximum frequency (3.074GHz). Then, results for a DVFS scenario (without ICARO) are shown. Finally, results for the three ICARO-DVFS versions are shown. It is worth recalling that the main ICARO goal is to keep background traffic unaffected when dealing with congestion-prone traffic, such as the simulated hotspot traffic. For that reason results for background and hotspot traffic are shown separately.

Figure 6.10 shows average network latency results. For the DVFS cases some peaks in latency can be seen when congestion background starts and ends. These peaks clearly show the overhead derived from the VR taking some time to perform the DVFS level changes. It is worth mentioning that the first peak is higher as the VR takes more time to change from the lowest frequency to the highest. After some analysis, we decided to increase from the minimum frequency to the maximum one since this involves only one transition, incurring much less penalty with respect to a step-by-step increase. As expected, the scenario with DVFS improves in power consumption the no-DVFS scenario, but at the cost of increasing network latency (Figure 6.12) and decreasing throughput (Figure 6.13). However, according to Figure 6.5, power consumption saving is more significant than performance degradation.

As can be seen in Figures 6.10, ICARO proposals separate effectively background traffic from the hotspot one, preventing the HoL-blocking effect over the former. As expected, ICARO-2VN achieves the highest improvements in network latency (up to 43%) since it takes into account all VNs to trigger the frequency-increment mechanism, and it is able to increment frequency to the maximum, but at the cost of increasing power consumption (8%). Nevertheless, despite being the ICARO proposal with the highest

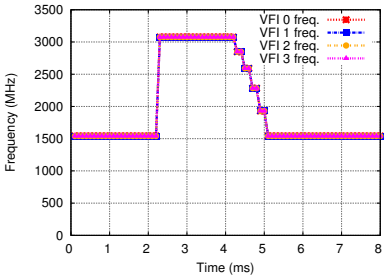


FIGURE 6.6: VFIs frequencies for DVFS without ICARO.

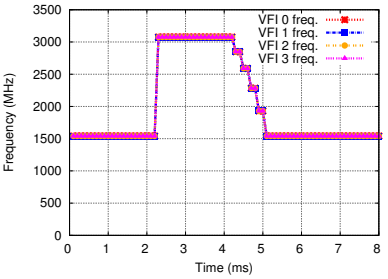


FIGURE 6.7: VFIs frequencies for ICARO-2VN.

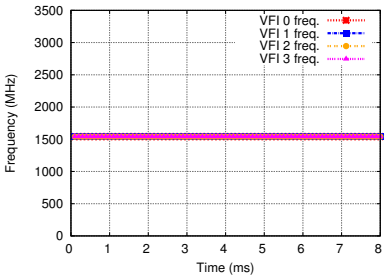


FIGURE 6.8: VFIs frequencies for ICARO-1VN.

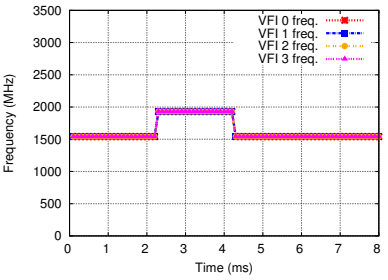


FIGURE 6.9: VFIs frequencies for ICARO-2GHz.

power consumption, this case still keeps power consumption below "DVFS-alone" levels while improving latency.

Regarding the results for ICARO-1VN, note that only *regular-VNs* (VN 0) is taken into account to increase the VFI frequency and only background traffic is forwarded through this VN, so that this slight traffic flow is not enough to trigger the frequency-increment mechanism and all VFIs end up working at the minimum frequency. Despite running at the minimum frequency, we can see that the background network latency is kept in similar values to the DVFS case working at highest frequency. This is achieved by separating both traffic types, so preventing the HoL-blocking that the hotspot traffic could cause. In addition, as frequency is the lowest allowed, power saving is maximum, achieving an improvement of 26%. However, as the system is working at the minimum frequency, throughput is lower than other cases that are running at higher frequencies. Nevertheless, background traffic achieves acceptable latency values.

Finally, ICARO-2GHz case could be considered as the best trade-off proposal. It takes into account all the available VNs but it is only allowed to increment frequency to the next step from the lowest frequency. This proposal does not achieve the best results in network latency and throughput, but nevertheless it improves the "DVFS-alone" case by 19% with a significant power-saving improvement (20%) with respect to DVFS, due to the lower frequency used.

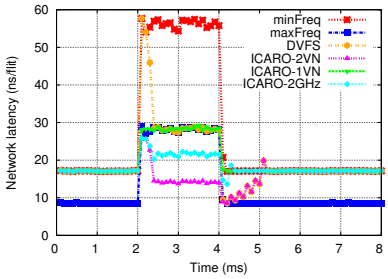


FIGURE 6.10: Network latency for background traffic.

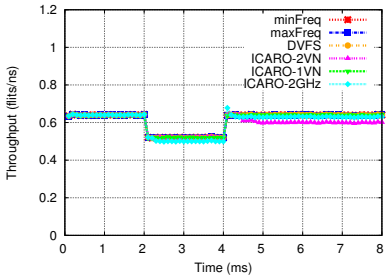


FIGURE 6.11: Throughput for background traffic.

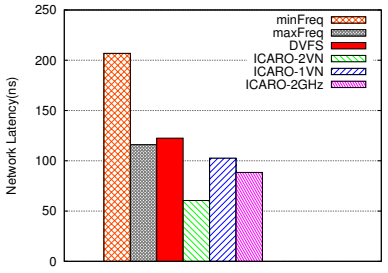


FIGURE 6.12: Final net. latency (all traffic).

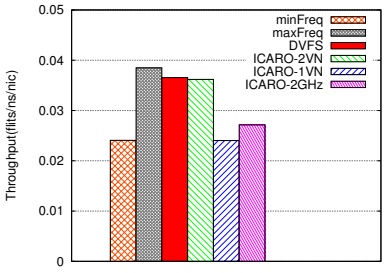


FIGURE 6.13: Final throughput (all traffic).

6.5 Conclusions and Future Work

In this work a combination of DVFS scheme with a congestion management mechanism (ICARO) based on separating harmful traffic from non-harmful one has been presented. Using the dedicated network used by ICARO in order to also deliver DVFS metrics for triggering frequency changes has been proposed. Three different approaches have been proposed in order to combine DVFS with ICARO. The first taking into account all VNs to trigger the frequency change, achieving the best latency improvements with a slight power consumption penalty. The second one only takes into account the *regular-VNs* (VN 0) to trigger frequency changes, achieving a power consumption improvement of 26%. Finally, the third proposal limit the frequency increase until ∼2GHz (instead of ∼3GHz), with a latency improvement of 18% and a power-saving gain of 20%. As future work we plan to use messages latencies as congestion metric and implement support in-order delivery support.

6.6 Acknowledgements

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2012-38341-C04-01 and by Ayudas para Primeros Proyectos de Investigación from Universitat Politècnica de València under grant ref. 2370.

Chapter 7

Increasing the Efficiency of Latency-Driven DVFS with a Smart NoC Congestion Management Strategy

- **Authors:** José Vicente Escamilla (Universitat Politècnica de València), Mario R. Casu (Politecnico di Torino) and José Flich (Universitat Politècnica de València)
- **Type:** Conference
- **Conference:** 10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)
- **Location:** Lyon, France
- **Year:** 2016
- **DOI:** 10.1007/978-3-319-27308-2_28
- **URL:** http://ieeexplore.ieee.org/document/7774444/
- **Citation:** [87]

7.1 Abstract

Dynamic Voltage and Frequency Scaling (DVFS) can be a very effective power management strategy not only for on-chip processing elements but also for the network-on-chip (NoC). In this paper we propose a new approach to DVFS in NoC, which combines a congestion management strategy with a feedback-loop controller. The controller sets frequency and voltage to the lowest values that keep the NoC latency below a predetermined threshold. To cope with burstiness and hotspot patterns, which may lead the controller to overdrive the NoC with too high frequencies and voltages, leading to excessive power consumption, the congestion management strategy promptly identifies the flows that caused the abnormal traffic situation and eliminates them from the latency calculation, leading to a significantly higher power saving. Compared to a baseline DVFS strategy without congestion management, our results show that our proposal saves up to 53% more power when bursty or hotspot-based traffic patterns are detected. In addition, since we also apply power-gating to make an efficient use of the network buffers, we achieve an improvement of up to 38% in power savings when no bursts or hotspots are present.

7.2 Introduction

Integrating a large number of processing elements into a single chip (CMPs, MPSoCs) is becoming the standard design choice in industry. This strategy offers good performance/power tradeoff while saves costs. These systems must be delivered with built-in networks, known as network-on-chip (NoCs) [86]. Usually, the NoC is designed with strict requirements in terms of throughput and latency so it becomes one of the most important chip components to guarantee the expected chip performance. In addition, the NoC may represent up to 20% of the overall chip power consumption [88].

The current trend is to increase the number of processing units as long as technology shrinks. Examples of this trend are the 72-cores Tile-Gx [7] or the 256-cores Kalray MPPA-256 (Bostan) [78]. With the number of processing elements increasing, it is mandatory to use strategies that reduce overall power consumption without affecting significantly system performance. One of the most successful mechanism to perform this is DVFS [3], which drives voltage and frequency dynamically at runtime to fit the workload requirements.

DVFS-based mechanisms essentially collect metrics from the system to find out how it is performing. According to this, the system reacts by increasing or decreasing frequency and voltage to meet the system requirements, thus saving power when requirements are low. The application of this mechanism to the NoC is not trivial. One main issue is to find the frequency that fits the whole NoC requirements. For small systems or systems

with a very regular workload, it may become trivial. However, in larger or non-balanced workloads, that target may be difficult to achieve since some parts of the network may be overloaded while the rest of the network is completely underutilized.

On the other hand, an emerging trend is, instead of using several identical cores, to manufacture heterogeneous chips [12]. This paradigm is based on the fact that specialized processing units are more efficient at performing specific jobs. However, due to this heterogeneity the network load becomes very unbalanced and unpredictable, characterized by hotspots-based traffic patterns [89]. In addition, some application traffic patterns may naturally generate abrupt traffic bursts [11] and generate congested network regions.

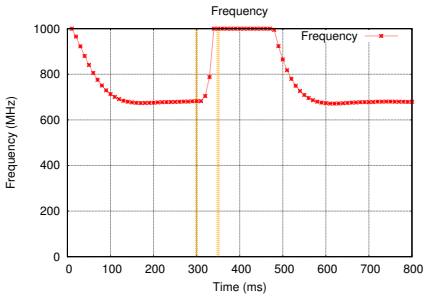


FIGURE 7.1: Frequency for DMSD under hotspot traffic.

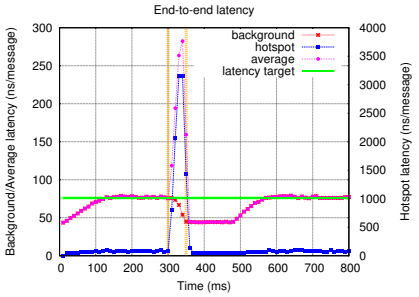


FIGURE 7.2: End-to-end latency per traffic type for DMSD under hotspot traffic.

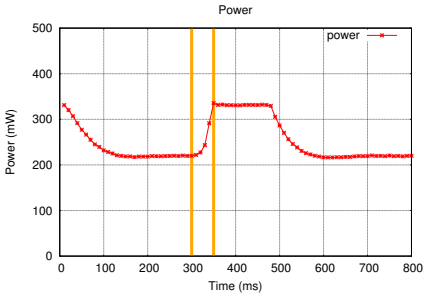


FIGURE 7.3: Power consumption for DMSD under hotspot traffic.

Because of these reasons, it becomes apparent that guaranteeing acceptable performance levels while reducing power consumption is a real challenge. To illustrate this issue, in Figs. 7.1-7.3 we can see how a system that uses the NoC latency as metric to drive the DVFS mechanism (DMSD) [54] behaves under a critical traffic pattern. This pattern mixes a *background* traffic generated by regular nodes in a 8x8 2D mesh (e.g. general purpose processing units) with a *hotspot* traffic injected by four nodes towards a single node. This hotspot is representative of traffic generated by device hardware accelerators [80] or irregular traffic generated by some applications [11], both characterized by short and heavy-weight data bursts from/to neighbor nodes. Specifically, the background traffic is generated and received by all nodes not belonging to the hotspot following an uniform distribution pattern. Accordingly, the hotspot traffic is generated by injecting traffic at a high data rate to a given node from all of its neighbors.

The results reported in Figs. 7.1-7.3 have been obtained with a cycle-accurate NoC simulator which models 4-stages pipeline routers: IB (input buffer), RT (routing), VA/SA (virtual channel and switch allocation), X (link crossing). In Tab. 7.1 the rest of configuration parameters are described. The values in Tab. 7.1 are used to obtain our baseline results[1]. To obtain our power results we used a modified version of Orion v3.0 [57].

Fig. 7.1 shows the frequency increase as the hotspot is activated at $300\,\mu s$. Fig. 7.2 shows how this increase differently affects the latency of two different traffic classes: congested traffic (hotspot traffic) and regular traffic (background traffic). The regular traffic is unnecessarily accelerated and its latency becomes less than a predetermined target (highlighted with a horizontal green dashed line), whereas the latency of the congested traffic increases significantly in spite of the high NoC clock frequency. As shown in Fig. 7.3, the consequence is a net power waste for an unwanted decrease of latency of the real productive traffic (the background one in our example).

In this paper we tackle such problem. We combine a latency-driven DVFS strategy (DMSD, as proposed in [54]), with a congestion management mechanism (ICARO [77]). Our goal is to use the congestion management strategy to discriminate and separate both traffic types, allowing the network to apply frequency and voltage policies based on the real productive traffic, hence allowing the DVFS strategy to guarantee a given end-to-end latency while optimizing power consumption.

The paper is organized as follows. First, we briefly describe DMSD and the methodology we used to obtain our results. Then, we describe ICARO. After presenting the combined strategy and its internal arrangement, we show the results. Then we revisit the related work. Finally we plot the conclusions and the future work plans.

7.3 Analysis of the DMSD DVFS Policy

The purpose of the *Delay-based Max Slow Down (DMSD)* DVFS policy is to decrease the NoC frequency and voltage as much as possible without compromising the system performance [54]. To achieve this, DMSD uses the average end-to-end latency as a performance metric, and a Proportional-Integral (PI) controller that adapts frequency and voltage so as to keep that metric close to a *latency target*. The higher the latency target, the larger is the power saving. In our experiments, we set it equal to the latency that is obtained with an injection rate 5% less than the saturation point under uniform traffic, which is obtained by running simulations in which the injection rate is increased until saturation. However, in a real system the latency target can be obtained by means of profiling.

[1] We also evaluated the robustness of our solution when some of these parameters are varied, as we will see in Sec. 7.4.5.

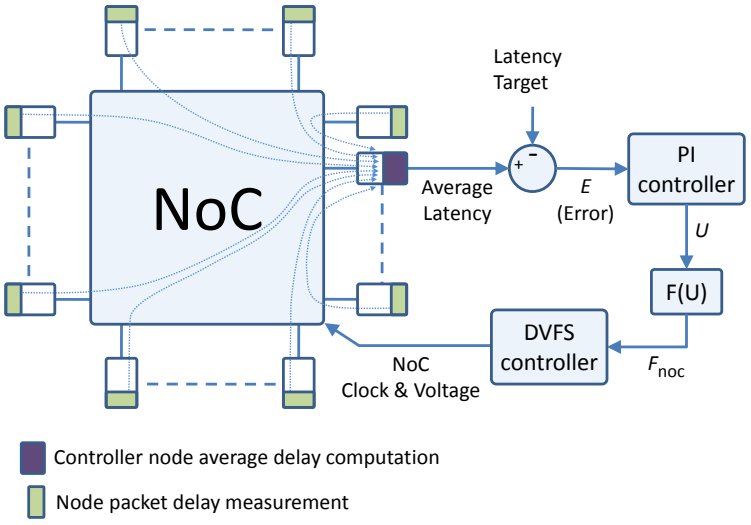


FIGURE 7.4: All nodes in the network send latency measures to the PI controller to set the new frequency.

In Fig. 7.4 an overview of an NoC provided with DMSD is depicted. Each node stores in a register the average end-to-end latency, updated each time a flit is received. Periodically, all nodes send the average latency to a given node, which computes the overall end-to-end latency for the whole system. In addition, this node contains the PI controller and the voltage and frequency controllers. Upon receiving all latencies, the overall latency at time n, L_n, is computed and the noise filter described by (7.1) is applied to obtain L'_n. Then, the *error* E_n is computed by subtracting the *latency target* L_t from L'_n as shown in (7.2). The error is then passed to the PI controller, which generates the signal U_n according to (7.3).

$$L'_n = \alpha\, L'_{n-1} + (1-\alpha)L_n \tag{7.1}$$

$$E_n = L'_n - L_t \tag{7.2}$$

$$U_n = U_{n-1} + K_I E_n + K_P(E_n - E_{n-1}) \tag{7.3}$$

In (7.3), K_I and K_P are the integral and proportional gains determined empirically and used to adjust the PI controller behavior while guaranteeing stability.

Finally, U is used to determine the frequency. For this, U is bounded within $U_{min_{sat}}$ and $U_{max_{sat}}$ and the range from U_{min} to U_{max} is linearly translated to frequency, as shown in Fig. 7.5. A voltage-to-frequency mapping is then used to apply the correct voltage for a given clock frequency.

DMSD performs well under stationary traffic patterns [54]. As shown in Figs. 7.1-7.3, however, the high intensity of a few data flows (hotspots) which are not representative

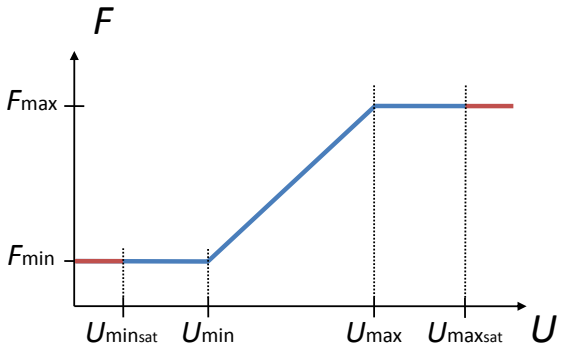


FIGURE 7.5: Conversion from U to frequency.

TABLE 7.1: Robustness analysis scenarios configuration.

Network configuration	
Topology	8x8 2D regular mesh
Routing policy	XY
Switching technique	Wormhole (flit-level)
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4 per Virtual Network
DMSD configuration	
Frequency range	333 - 1000 MHz
Voltage range	[0.56, 0.9] V
K_i, K_p	0.025, 0.0125
U saturation range	[-15, 15]
α	0.7

of the whole system load, disrupts the DVFS strategy, leading to a waste of power by increasing the frequency and voltage unnecessarily.

Notice that this effect could be avoided by implementing Voltage and Frequency Islands (VFIs) [55][56]. However, this would imply extra silicon area and power to implement the VFIs separate DVFS controllers, and it would require to either know at design-time where hotspots will be located, or the ability to confine the hotspot traffic in a separate voltage island at run-time. In contrast, our approach consists in implementing a congestion control mechanism (ICARO) to detect hotspots and filter them out, regardless their location and intensities.

Orion does not directly support the industrial 28-nm CMOS technology that we used for the implementation of routers with support for congestion management and buffer power-gating. By using the post-synthesis results of our RTL version of the router, we modified Orion in such a way that its results are compatible with our technology.

Moreover, we added the support for including the effect of buffer power-gating in the computation of power consumption.

7.4 Implementing Congestion Management

Hotspot flows mask the overall system performance, increasing significantly the overall latency while the network resources not used by the hotspot flows may be underutilized. Our approach consists in identifying those hotspot flows, and separating them from the rest of the network traffic (background traffic). For this purpose, we pick ICARO, a congestion-control mechanism that detects, identifies, and isolates congestion within the network.

7.4.1 ICARO

ICARO removes the Head-of-Line (HoL) blocking by first identifying congestion, and then by isolating the congested flows involved in it into dedicated Virtual Networks (VNs). As the congested traffic is delivered through separate resources, it does not share buffer resources with the non-congested traffic, so HoL-blocking is removed. ICARO consists of three stages: congestion detection, notification and isolation.

7.4.1.1 Congestion Detection

The congestion is detected at the routers level, by keeping track of which input ports are requesting any output port. If more than one input port is requesting a given output port for too much time, that output port is marked as oversubscribed, so congestion at that output port is declared. However, two or more input ports could be requesting a given output port at a low data rate, not leading to congestion. Because of this, only input ports exceeding a given utilization threshold are considered.

7.4.1.2 Congestion Notification

Once congestion is detected, it must be notified to the NIs to isolate the traffic that causes the congestion before being injected into the network. To perform this, ICARO uses a dedicated congestion notification network (named *CaL* network[2]), which consists of a ring of N registers connected by links of $log_2(N) + p + 1$ width, where N is the number of nodes and p the routers radix. An example of a CaL network is shown in Fig. 7.6. Other mechanisms for fast notification delivery include *express channels* [90]

[2]This network is called CNN in [77] but we modify it to deliver also other messages, hence the name CaL (Congestion and Latency) network.

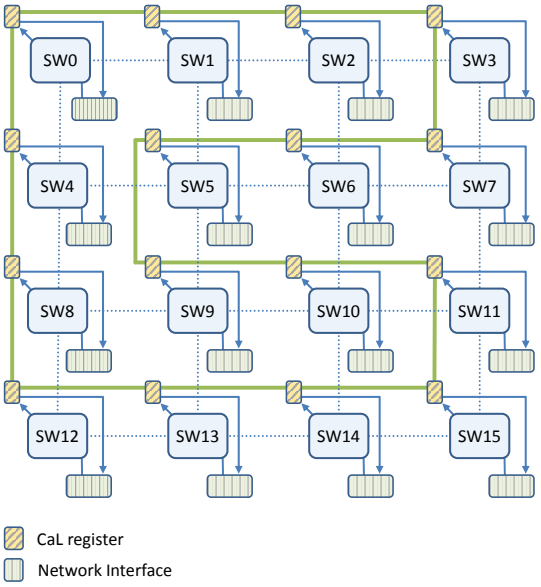


FIGURE 7.6: Congestion Notification Network for a 4x4 mesh.

and circuit-switched networks. However, express channels may imply high area overhead while circuit-switched networks suffer from high latency penalties when establishing circuits.

7.4.1.3 Congestion Isolation

In absence of congestion, the NI allocates all messages in *regular-VNs*, as determined by the NI allocator. Once notifications are received, the NI uses the congestion information and calculates a message route to know whether that message will cross any of the *congested points (CPs)*. If so, the message is reallocated into a special VN (*extra-VN*) to be injected and delivered through it along all the path to destination. Otherwise, the message is injected through the current regular-VN. By doing this, flows contributing to the congestion are isolated into the *extra-VN*, keeping the non-congested traffic into the *regular-VNs*. In this way, DMSD will be able to measure the latency of the non-congested traffic (data flowing through the *regular-VNs*). Fig. 7.7 depicts an example of this process for the NI 4 in a 4x4 2D mesh.

When congestion ends, the conditions leading to detect CPs at routers will not occur anymore. Therefore, all the routers that previously detected CPs will detect this end of congestion and notify this event to the NIs, which will react by removing those CPs from their congestion notification board. Accordingly, since the messages pending to be injected at NIs will not cross any stored CP, those messages will be normally injected through the *regular-VN*.

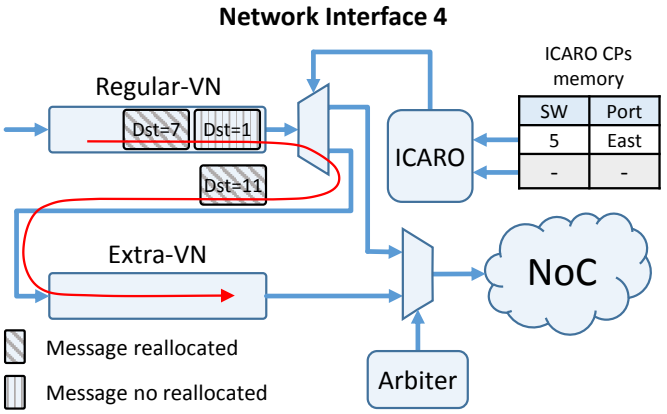


FIGURE 7.7: NI with ICARO for reallocating congested messages.

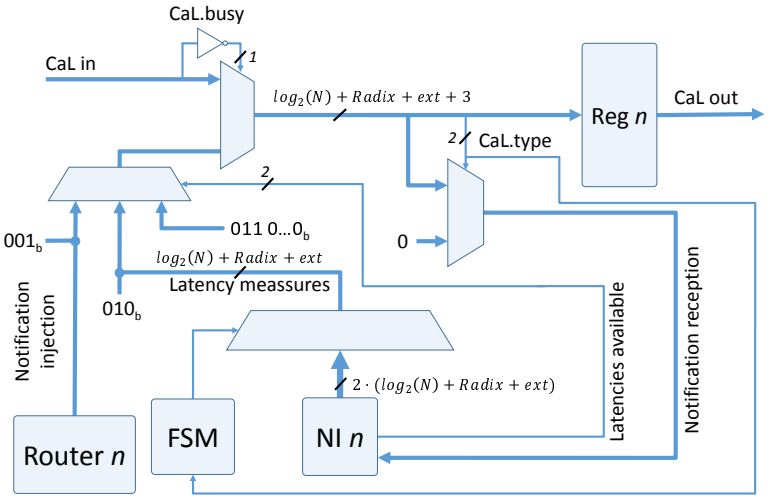


FIGURE 7.8: CaL network register associated logic for regular nodes adapted to DMSD.

7.4.2 Delivering Latency Measurements with the CaL Network

In the original DMSD formulation, packets containing the measured end-to-end latency are sent to the controller node via piggybacking [54]. Intense congestion situations, however, may delay the delivery of those packets and the reaction of the PI controller, potentially causing the PI controller to oscillate. On the other hand, ICARO implements the CaL dedicated network, which we modified to support the delivery of those latency values in addition to congestion notifications. By doing this, the metrics necessary to set the frequency properly are guaranteed to timely arrive at destination (with low aggregated overheads, as we show in Sec. 7.4.4).

In a DMSD-based system there are two types of nodes: those that send their latency metrics, and one that receives them. Thus, we implement two slightly different logic blocks to connect to the CaL network. Fig. 7.8 shows the logic of a typical router/NI that sends and receives ICARO notifications and sends DMSD latencies. Note that, despite ICARO notifications are sent in one cycle, we modified the logic to serialize

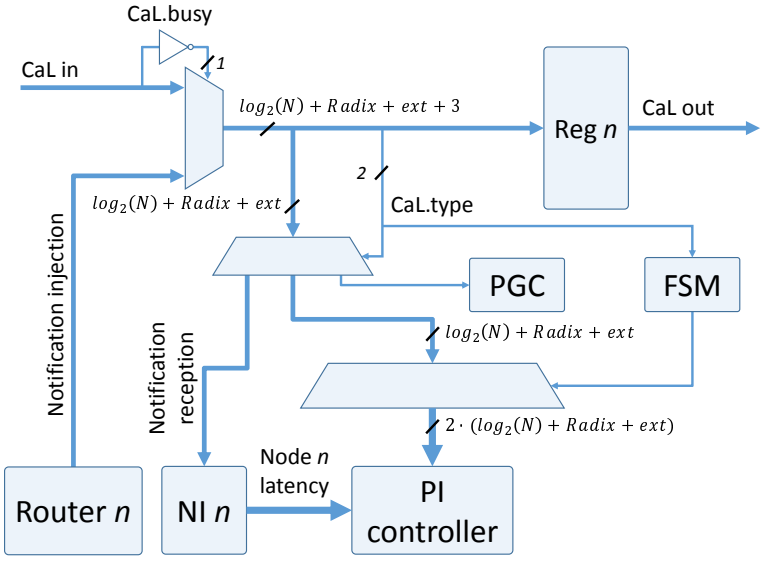


FIGURE 7.9: CaL network register associated logic for the node provided with the PI/DVFS controller adapted to DMSD.

the transmission of 32-bit latencies through the CaL network by extending its links width by (*ext=8-Radix*) bits. Congestion notifications can be seamlessly interleaved with DMSD latency notifications because one bit identifies the type of CaL message. DMSD notifications from different nodes are guaranteed to arrive in order since nodes will send them after a fixed (and different for each node) time offset. Similarly to the sender node in Fig. 7.8, Fig. 7.9 shows the logic for the receiver node. This node sends and receives ICARO notifications like any other CaL node, adds its own latency measurements to the received ones, and forwards them to the PI controller.

7.4.3 Power-Gating Extra-VN Buffers

To avoid wasting power when there is no congestion, we implement a mechanism to power-off the extra-VN buffers via a centralized Power-Gating Controller (PGC), which resides in the same node that implements DMSD. All the buffers of an extra-VN (in all NIs as well as in all routers) are powered on/off simultaneously. Power-on is easy: the PGC node snoops the CaL network and when it catches the first congestion notification it broadcasts a power-on message through the CaL network. On the contrary, power-off is not trivial. Snooping the CaL network in search of the "end of congestion" messages is not a valid strategy because there might still be messages in the extra-VN, either in the NIs pending to be injected, in the routers on their way to destination, or both. Therefore, to safely turn off the extra VNs, the PGC must be informed through the CaL network by both all NIs and routers about their detection of a *congestion-free* situation:

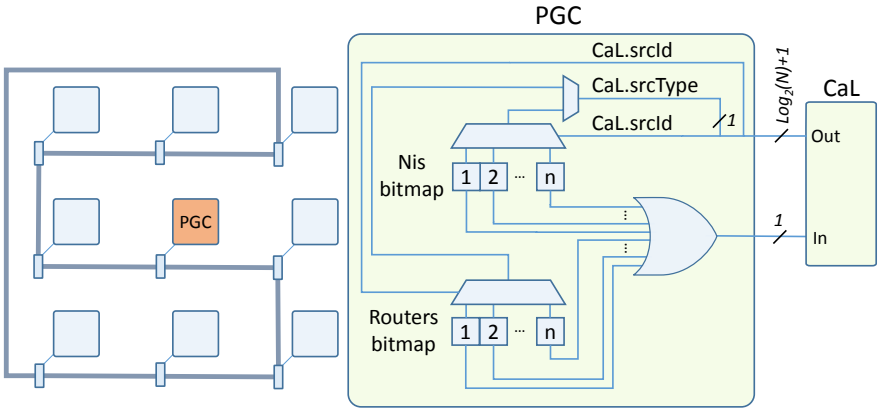


FIGURE 7.10: Power-gating controller.

7.4.3.1 Network Interfaces Detection

To make sure that no congested traffic will be injected into the network from a given NI, two conditions must be satisfied. First, the extra-VN buffers must be empty. In addition, the NI congestion notification board must be clean (no new notifications). If both conditions are met, the NI notifies its *congestion-free* status to the PGC with a special message sent through the CaL network.

7.4.3.2 Routers Detection

At routers the mechanism is simpler. Each time the *extra-VN* buffer utilization increases from 0 to 1, the router sends a message to the PGC to inform that is storing congested traffic. On the other hand, when the buffer utilization decreases from 1 to 0, the router sends another message to inform that is *congestion-free*.

The PGC is provided with two N-width bitmaps, where N is the number of nodes in the network: one bitmap for the NIs and one for the routers. These bitmaps are updated any time a NI or a router notifies the PGC about its status (0=no congested traffic stored, 1=congested traffic stored). In this way, the PGC has a complete *congestion picture* of the network. When the PGC detects that all NIs and routers are *congestion-free*, it commands to power-off the extra-VN buffers; otherwise, it commands to power-on the buffers. Fig. 7.10 sketches the PGC bitmaps, the logic to power-on/off the buffers, and its connection to the CaL network. We quantify the advantage of using power-gating in Sec. 7.4.5.

Note that area and power consumed by the PGC are negligible since its implementation only requires N-width demultiplexers, 1 N-width multiplexer, 1 N-width NOR gate and $2N$ registers for a complete mesh.

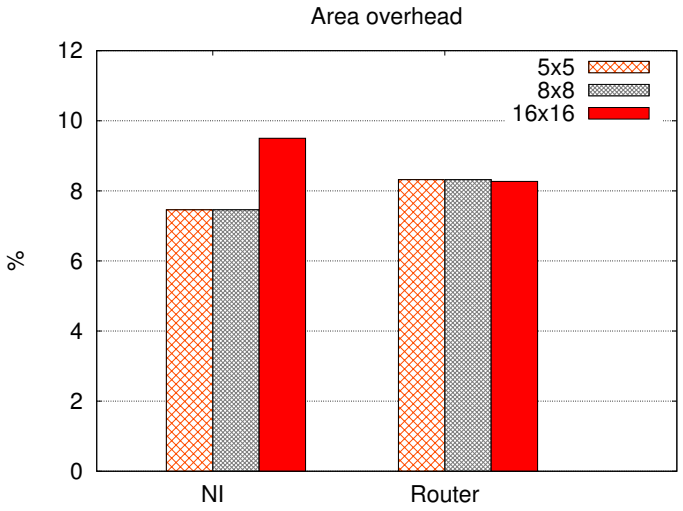


FIGURE 7.11: ICARO-DMSD area overhead of different meshes.

We are aware that turning on/off all the *extra-VN* buffers is suboptimal since several buffers could not be reached by any congested flow. Because of this, as a future work we plan to implement a new policy to turn the buffers on/off selectively.

7.4.4 Area Overhead Analysis

The bars in Fig. 7.11 illustrate the area overhead for a NI and a router with support for ICARO, with respect to a baseline implementation (no DMSD, no ICARO)[3]. The results have been obtained after synthesis on our 28-nm technology, in the conditions of Tab. 7.1, except for the mesh size that we let vary. We notice that the overhead is small, less than 10%, even for the case of a large 16×16 mesh.

7.4.5 Experimental Results

In this section we first report simulation results obtained in the baseline configuration of Tab. 7.1. These results show that our combined DVFS and congestion management strategies can effectively solve the problem outlined in Sec. 7.2 that is at the basis of our work. Then, we report results obtained with a sensitivity analysis in which we varied several configuration parameters to check the robustness of our solution. Note that, for our experiments DMSD as well as ICARO are provided with the same amount of VNs in order to compare both solutions with the same amount of resources, providing each VN with the same amount of VCs. However, since DMSD does not require several

[3]We obtained overhead results only for ICARO since DMSD and the PG controller overheads are negligible compared to the ICARO's overhead.

VNs to work properly and these additional resources may affect negatively to its power consumption we perform an additional experiment comparing against the baseline with only 1 VN.

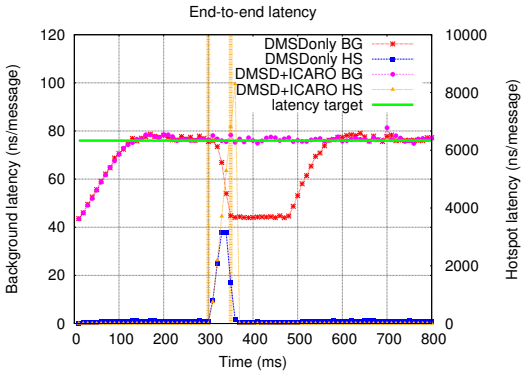


FIGURE 7.12: End-to-end latencies for the background and the hotspot traffic.

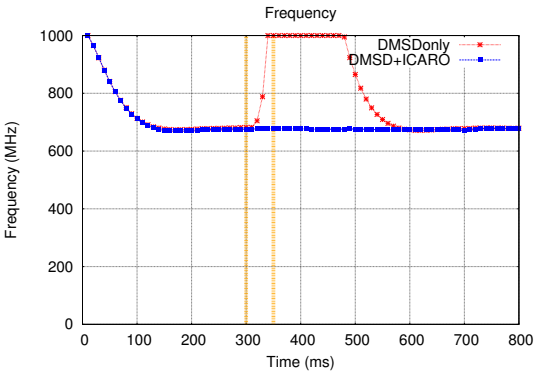


FIGURE 7.13: Frequencies for DMSD and ICARO-DMSD.

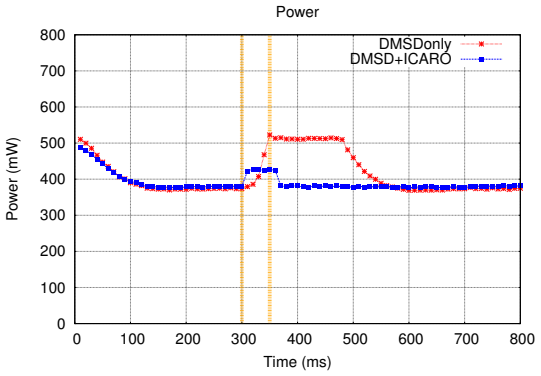


FIGURE 7.14: Power consumption for DMSD and ICARO-DMSD.

Figs. 7.12-7.14 compare the *DMSD* and the *ICARO-DMSD* cases in terms of latency, frequency, and power, in the baseline scenario. Notice that to properly compare the two cases, the two systems have the same total buffering resources. Note also that, in the case of ICARO, the *extra-VN* is composed of as many VCs as the *regular-VNs*.

Since ICARO effectively separates the background traffic from the hotspot one, DMSD can effectively measure only the latency of the background traffic. Therefore, thanks to the PI controller, DMSD keeps the latency of the background traffic around the 76-ns *latency target*, as shown in Fig. 7.12. In fact, as Fig. 7.13 shows, the NoC clock frequency is not influenced anymore by the activation of the hotspot traffic. This, in addition to the use of power-gating, results in a significant improvement of the power consumption, as shown in Fig. 7.14. When the hotspot is not active (from time 0 μs to 300 μs, and then again after around 380 μs), the extra-VN buffers are powered-off, resulting in lower power for the *ICARO-DMSD* case. When the hotspot is active, the extra-VN buffers are

TABLE 7.2: Robustness analysis scenarios configuration.

Scenarios							
Label	Mesh Size *(nodes)*	Queue Size *(flits)*	VCs	Msg. Length *(flits)*	Num. HS	HS Dur. *(ns)*	Lat. target *(ns)*
Baseline	8x8	4	4	10	1	50us	76
5x5	5x5	4	4	10	1	50us	66
16x16	16x16	4	4	10	1	50us	105
qs2	8x8	2	4	10	1	50us	79
qs8	8x8	8	4	10	1	50us	81
qs16	8x8	16	4	10	1	50us	72
vcs2	8x8	4	2	10	1	50us	60
vcs8	8x8	4	8	10	1	50us	97
ml5	8x8	4	4	5	1	50us	62
ml20	8x8	4	4	20	1	50us	96
2HS	8x8	4	4	10	2	50us	76
3HS	8x8	4	4	10	3	50us	76
short	8x8	4	4	10	1	25us	76
large	8x8	4	4	10	1	100us	76

switched on, hence the power increases. Still, since the clock frequency in the *ICARO-DMSD* case is less than the *DMSD* case, the power consumption is also significantly reduced.

To validate our results under different network configurations, we changed several network parameters: mesh size, router buffers queues size, number of virtual channels, message length, number of hotspots, and hotspot duration. All the cases analyzed are described in Tab. 7.2, in which every case is assigned a label that is used next in the graph keys. As Fig. 7.15 shows for all the configurations analyzed, in the *ICARO-DMSD* case the background traffic correctly tracks the prescribed target, hence avoiding the excessive power consumption that characterizes the reference *DMSD* case. Note that, since the goal of our approach is to keep the background latency around the latency target, for better understanding, hotspot latencies have been omitted in the graphs. Also note that the hotspot start/stop time is highlighted with vertical bars and that in the *hotspot duration* graph the three different hotspot ending times are highlighted with different colors. Please note that the *latency target* value for a given scenario depends not only on the saturation point, which is highly correlated with the system configuration, but also on the latency curve gradient. Therefore, in some system configurations the calculated *latency target* seems not to follow an intuitive progression like in the VCs analysis graph shown in Fig. 7.15.

Fig. 7.16 summarizes the improvement of power consumption of the *ICARO-DMSD* case, in all the configurations of Tab. 7.2. Two different improvement values are reported.

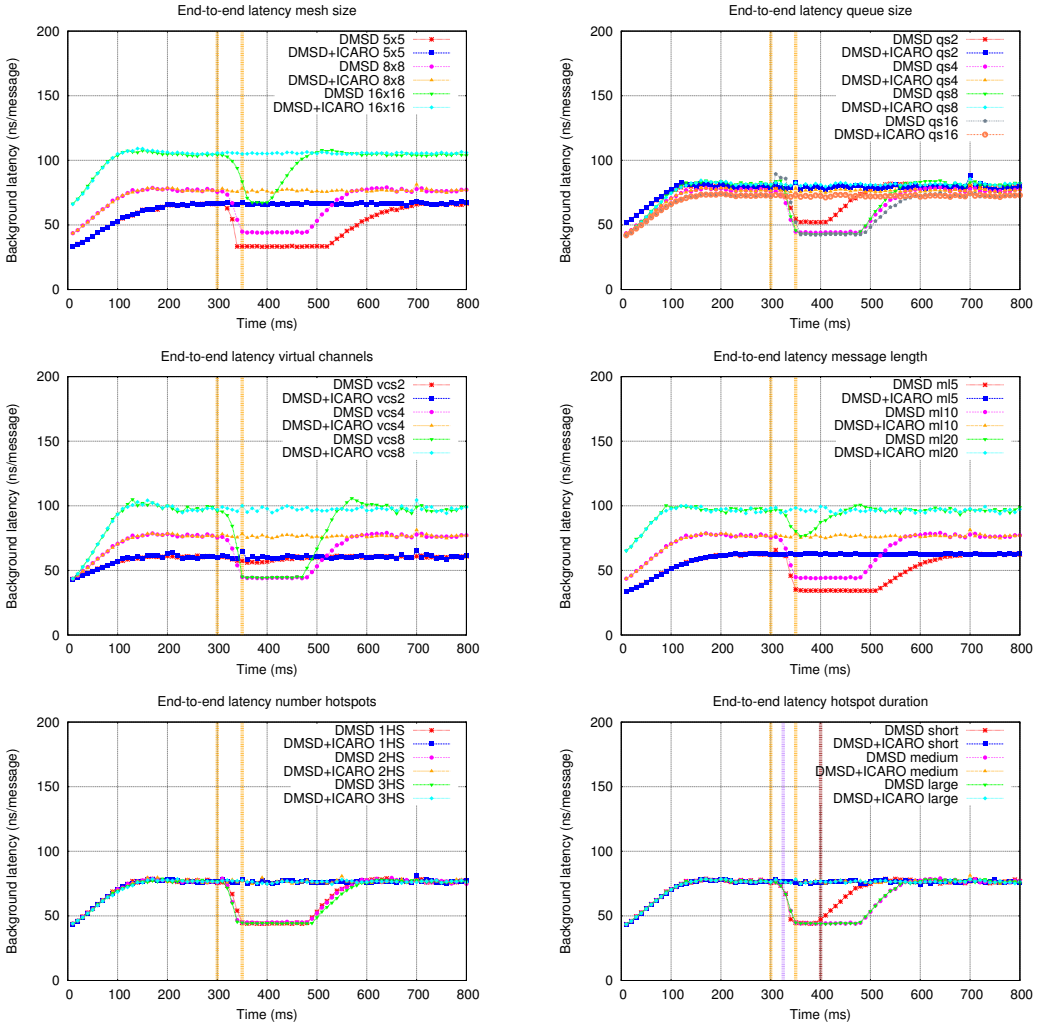


FIGURE 7.15: End-to-end latency for different configuration parameters

The first one is due to the *extra-VN* power-gating (*no-HS* in the graph), measured at time 290 μs (just before the hotspot activation); the second one corresponds to the power-saving during the hotspot duration (*HS* in the graph) and is calculated by averaging the power spent from time 300 μs to time 600 μs, since this is the time range in which the hotspots affect any of the cases analyzed. Note that the power overhead due to the additional hardware required by our proposal is already included in the power consumption graphs.

As Fig. 7.16 shows, for all the cases considered, the combination of DMSD and ICARO leads to a significant power improvement over the DMSD baseline when hotspot is active. When no hotspot is active, by switching the *extra-VN* off we achieve up to 38% power saving and an average of 28%. When congested traffic is detected, ICARO manages this sort of traffic and DMSD tunes the frequency properly saving up to 53% power consumption and 38% on average. In the results obtained when the hotspot is present, we observe a larger variance. This is expected as, for calculating the average, we take

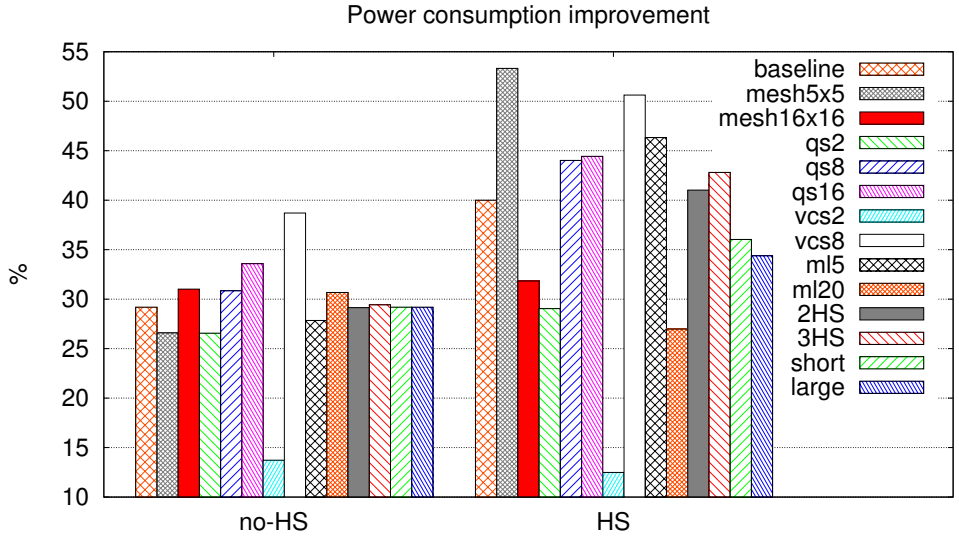


FIGURE 7.16: Power consumption improvement with respect to DMSD for all configurations.

values from the same range of time for all cases but the duration of the effects of the hotspots are not the same for those cases, therefore, the weight of those values over the average is not the same.

In the final experiments we analyze our proposal against DMSD provided with 1VN. Unlike ICARO, DMSD does not require several VNs to perform properly, so we performed the same robustness analysis shown above but providing DMSD with 1VN. Nonetheless, as ICARO does not require the *extra-VN* to be provided with several VCs, for this simulations we configured ICARO with the same number of VCs for the *regular-VN* as the DMSD case and only 1VC for the *extra-VN*. The power results in Fig. 7.17 show that in absence of congestion, ICARO consumes more power than DMSD due to the ICARO logic power consumption. When hotspot is active, however, despite the additional buffers ICARO saves a significant amount of power under the most part of the analysis, achieving up to 20% power saving. Nevertheless, some scenarios present characteristics (amount of resources, message length, etc.) for which DMSD does not overreact to keep the latency under the target, resulting in less power consumption for DMSD. Still, the congestion in those scenarios triggers the ICARO mechanism, causing to switch the extra-VN buffers on, increasing power consumption, and ultimately reducing the power saving compared to the baseline.

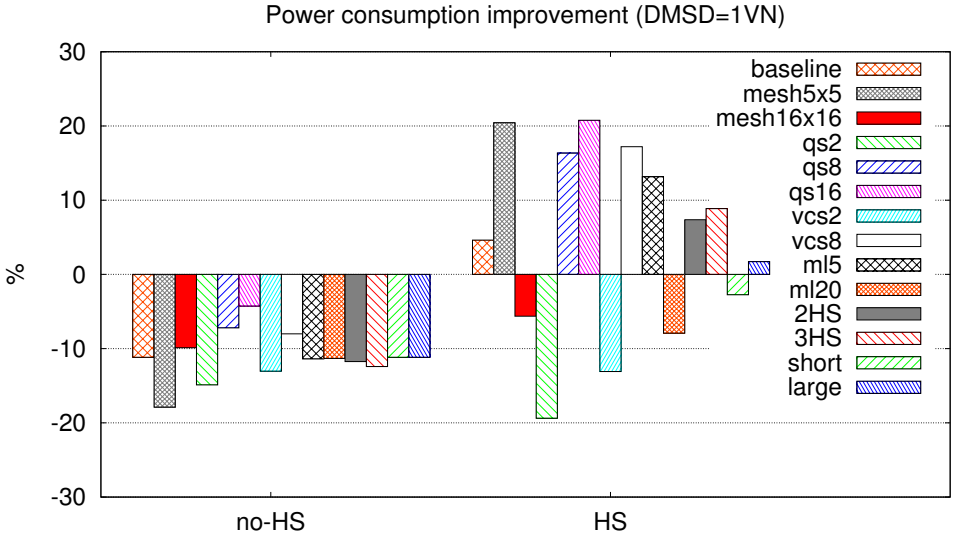


FIGURE 7.17: Power consumption improvement with respect to DMSD (provided with 1VN) for all configurations.

7.5 Related Work

Most of the literature focuses on a fine-grain application of DVFS to NoCs, with routers and even links individually powered at different voltages and frequencies [27][28][29][30][31][32]. These works, however, do not consider the overhead of having multiple voltage regulators and PLLs for the various NoC components, not to mention the latency penalty due to multiple clock-domain crossings. We share the view of other authors that consider more practical to have a single voltage and frequency domain for the whole NoC [34][35][36][37].

It is apparent that a fine-grain DVFS approach would lead to better power savings, but the implementation cost would be too high. For these reasons researchers explored a middle ground that we can classify as coarse-grain NoC DVFS, in which either multiple NoC planes (typically two planes) powered at different voltages and/or frequencies are used [38][39], or routers that can individually choose between only two voltages are employed [40]. Our approach can be easily adapted to the case of multiple NoC planes.

In terms of implementation of the DVFS controller, our work has features in common with [36], in which a PI-based DVFS is applied to the NoC and the last-level cache of a Chip Multi-Processor (CMP). A different approach to this problem is proposed in [37], in which the DVFS controller is based on an artificial neural network trained with the help of a PI controller. Differently from these works, we do not restrict our study to the CMP case and analyze the effect of hotspot traffic on the behavior of the PI-based DVFS controller.

Regarding congestion management, most of the solutions in the literature are based on monitoring congestion metrics and using them to make routing decisions. Following this paradigm, RCA [20] uses multiple global metrics collected from the whole network to select at each router the output port which messages are forwarded through. Differently from our approach, RCA collects metrics delivered through the regular network. Thus, if some metrics travel along already congested routes, this may slow down the metrics collection, causing the mechanism to make wrong decisions. Besides, adapting the routes to avoid hotspots may result in moving the location of such hotspots from one place to another. Finally, avoiding hotspots may be impossible if all the flows are bound to the same destination (e.g. the memory controller).

The authors of [24] propose HPRA, a hotspot-formation prediction mechanism. HPRA uses an Artificial Neural Network-based (ANN) hardware that gathers buffer utilization data to predict the formation of hotspots. Then, HPRA classifies the traffic into two classes: hotspot-destined traffic (HSD) and non-hotspot-destined traffic (nonHSD). HSD traffic is throttled at source while the nonHSD traffic is routed avoiding paths containing hotspots routers. However, in the cases in which the ANN fails to predict hotspots, it may redirect traffic to an unpredicted hotspot, causing an even worse degradation of the system performance. Besides, HPRA suffers from the same metrics delivering issue of previously described RCA.

In [91], the authors propose a mechanism to monitor the state of the network in order to select the best path to deliver each packet. To select the best next router for a given packet at each hop, each router in the network must know the best path to follow from the current node to the packet's destination. This requires sending back a special message with the route status information for each message sent from a given node to a given destination. This may cause a waste of network bandwidth and, in presence of several or sudden congestion, it may be affected by the same problem of delayed metrics delivery described before. In addition to this, this mechanism requires that each node keeps a table composed of one entry for each node in the network. This means that the total stored aggregated data in the whole network grows quadratically with the number of nodes, which clearly hampers scalability for large mesh sizes. In contrast, in our case every node only stores a 32-bit latency value, which results in a linear growth with the number of nodes.

7.6 Conclusions and Future Work

In this paper we present an integrated approach for saving power while guaranteeing latencies in NoCs under non-stationary traffic patterns. We demonstrate that by integrating a congestion management strategy and a loop-based DVFS controller to tune

the frequency for saving power while guaranteeing a latency target, we obtain a power-effective strategy. As our results show, we save up to 53% power compared with the baseline DVFS system in presence of hotspots. In addition to this, we propose a power-gating mechanism to power-off buffers when not needed, resulting in up to 38% power saving in absence of congested traffic. To obtain these results, our approach requires a small area overhead, less than 10%.

As future work we plan to compare the approach reported in this paper with another one that separates traffic classes using physically distinct networks with two different DVFS controllers rather than different virtual networks. In addition to this, as already mentioned, we plan to implement a smarter mechanism to power buffers on/off selectively to achieve best power saving results.

Acknowledgments

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2015-66972-C05-1-R and by Ayudas para Primeros Proyectos de Investigación from Universitat Politècnica de València under grant ref. 2370. We also want to thank especially the HiPEAC project that supported the internship during which this work was developed.

Chapter 8

ICARO-PAPM: Congestion Management with Selective Queue Power-Gating

- **Authors:** José Vicente Escamilla (Universitat Politècnica de València), José Flich (Universitat Politècnica de València) and Mario R. Casu (Politecnico di Torino)
- **Type:** Conference
- **Conference:** International Conference on High Performance Computing & Simulation (HPCS)
- **Location:** Genoa, Italy
- **Year:** 2017
- **DOI:** Pending
- **URL:** Pending
- **Citation:** Pending

8.1 Abstract

The growing demand for performance and technology advances drive manufacturers to integrate more and more cores in the same die. However, this increment of interconnected computing elements implies more pressure over the network-on-chip, which might saturate, leading to congestion and, thus, degrading system's performance. To deal with this, ICARO was recently proposed as a congestion control mechanism which identifies congested points and isolates congested traffic in separate queues, removing the HoL-blocking effect, hence, leaving congestion harmless. However, ICARO's additional buffers incur in significant power overhead. In this paper, we propose a new version of ICARO (ICARO-PAPM) which is integrated with a novel path-oriented fine-grained power-gating mechanism (PAPM). PAPM can selectively power on and off paths partially shared by different sources. When driven by ICARO, unused queues for congested traffic can be powered down, thus saving energy. We demonstrate that ICARO-PAPM does not interfere with the original ICARO performance, while it achieves a significant reduction of 35% in power consumption by keeping all additional buffers powered off when no congestion arises on the network, and up to 27% under congested traffic by powering on only those queues needed by the congested traffic.

8.2 Introduction

Chip MultiProcessors (CMPs) and MultiProcessors System on Chips (MPSoCs) are being designed and manufactured by the industry. Such designs offer good performance/power tradeoff while saving costs. The current trend is to implement more and more processing units motivated by the advances in the integration scale. Examples are the Tile-Gx [7] or the 256-cores Kalray MPPA-256 (Bostan) chip [78]. These processing units need to be interconnected by an on-chip-network [86], which is in charge of exchanging data between processing units or nodes. The network must be able to deliver the huge amount of data generated by all nodes subject to very tight requirements in terms of latency and throughput. Nevertheless, the constant increase in number of nodes brings also an increase in the amount of data to be delivered, driving the network to its performance limits, thus becoming critical to manage the network traffic properly in order to achieve the expected system performance.

Additionally, data produced by hardware accelerators (e.g.: cache prefetchers[79][80]), heterogeneous systems [12] and some specific applications, tend to generate irregular traffic patterns[11] or hotspots. This sort of traffic patterns may saturate some points in the network, generating congestion which might be easily propagated through the rest of nodes and generate HoL-blocking [14](HoL), degrading the overall system performance. HoL-blocking is given when flits stored in FIFO queues get blocked due to the lack

TABLE 8.1: Orion configuration parameters.

Orion configuration	
Technology	28nm
Vdd	0.9V
Frequency	1GHz
Toggle rate	0.25
VCs	4
Queues size	4 flits
Flit width	128 bits
Input/Output ports	5
Output buffers	0

of resources required by the flit at the head of the queue preventing other flits from advancing.

In this concern, ICARO was recently proposed [77]. ICARO is a congestion control mechanism that works by identifying congested flows and isolating them into an additional Virtual Network (VN) implemented in a separate queue in all the routers. By doing this, ICARO avoids interactions between congested and non-congested traffic, therefore removing the HoL-blocking effect.

On the other hand, the network size affects dramatically the network power consumption, representing up to 20% of the overall chip power consumption [88]. Therefore, new mechanisms to improve network performance must be carefully designed to be also power efficient. In this concern, there are different strategies that can be adopted to reduce network power consumption. One of the most extended strategies is to dynamically power off different parts of the network (*power-gating* [4]), like routers. Nevertheless, to power off a whole router implies important drawbacks, such as the need of building an strategy to support traffic rerouting and to guarantee deadlock-free condition. We performed an analysis of power consumption for each main component of a router using Orion v3.0 [57] with the configuration parameters shown in Table 8.1. Note that Orion does not originally support the 28-nm CMOS technology that we used for our implementation. By using the post-synthesis results of our RTL version of the router, we modified Orion in such a way that its results are compatible with our technology. Moreover, we added the support for including the effect of buffer power-gating in the computation of power consumption. As seen in Figure 8.1, the router buffers are the most power-hungry component of the router. Therefore, it makes sense to follow a buffer power-gating strategy, rather than a router power gating strategy. In this way, if we power a buffer off, we can still serve traffic without the need of a complex rerouting strategy.

In this paper we revisit ICARO in order to complement it with a new power-gating mechanism (PAPM) to alleviate the power overhead caused by the additional buffers used by ICARO for its additional VN. The new mechanism, ICARO-PAPM, will power

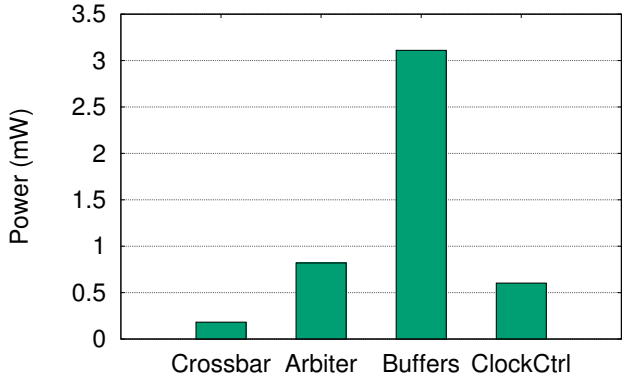


FIGURE 8.1: Router modules power consumption.

on only those queues needed to store congested traffic. In the absence of congestion, all the queues will be powered off.

The rest of the paper is structured according to the following. First, we describe ICARO. Next, our novel power-gating proposal (PAPM) is presented. Then, we show the experimental results by comparing ICARO and- ICARO-PAPM. Finally we present our conclusions and related work.

8.3 ICARO

ICARO is a congestion control mechanism that works by isolating congested traffic flows from non-congested ones by means of VNs. By isolating congested traffic, ICARO removes the HoL-blocking effect. ICARO implements an extra queue (referred to as *extra-VN*) on each input port of every router, and also on each end-node. This queue is used exclusively for packets contributing to congestion. ICARO functionality can be divided in three phases: congestion detection, notification and traffic isolation.

8.3.1 Congestion Detection

Congestion is detected in routers essentially by detecting when an output port is contended (requested by 2 or more input ports) during a relatively large amount of time (determined empirically). Once congestion is detected, the oversubscribed output port is notified as a $\{router, port\_status\}$ pair value called *CP* (Congested Point) to all the end-nodes. Indeed, each router periodically notifies to all end-nodes the current status of each output port (congested, non-congested).

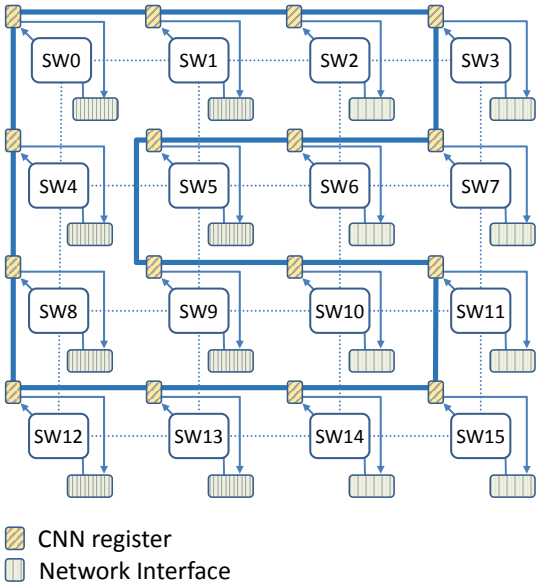


FIGURE 8.2: Congestion Notification Network for a 4x4 mesh.

8.3.2 Notification

The notification of a port being congested or not is delivered to all end-nodes through a light-weight separate network (Congestion Notification Network; CNN). The *CNN* network consists of a ring of N registers connected through unidirectional links of $log_2(N) + p + 1$ width, where N is the number of routers and p is the router radix. An example of a CNN network in a 4x4 mesh NoC can be seen in Figure 8.2. Once a notification arrives to an end-node, this notification is stored in a local memory cache at the NI. This memory is arranged in rows (e.g. 4 rows for a 8x8 mesh) so that each row is populated with one CP. Entries are dynamically allocated and deallocated based on the CP notifications received.

8.3.3 Isolation

ICARO uses two types of VNs: regular-VN and extra-VN. The first one is used for non-congested traffic and the second one for congested traffic. For each message to be injected into the network, the NI checks whether the message will cross an identified CP. If so, the message is reallocated into the extra-VN and will traverse the network by using only extra-VN queues. Otherwise, it will be injected through the regular-VN. Figure 8.3 shows an example of ICARO. Router 4 has detected a *CP* at the east port. Then, node 3 injects a message to node 6, which is not forwarded through the extra-VN since it does not cross any detected *CP*. However, it also injects a message to node 5, which will cross the east port of router 4. Therefore, this message is injected and forwarded through the extra-VN. As can be seen, the *CP* affects also to destination nodes 2 and

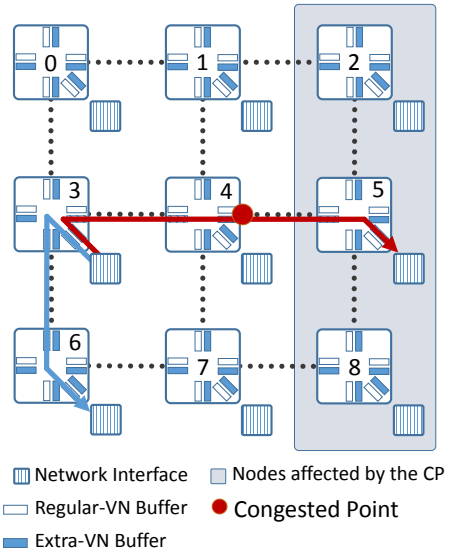


FIGURE 8.3: ICARO example of node 3 sending a congested message to node 6 and a non-congested one to node 5.

8 because both are reachable through the *CP* by source 3, hence, any message to those nodes will be forwarded through the extra-VN as well.

8.4 PAPM: Path Aware Power Mechanism

8.4.1 Overview

The goal of our proposal is to reduce power consumption in ICARO by implementing a new buffer power-gating mechanism (PAPM) for the *extra-VN* queues. Essentially, PAPM will power on only those queues which are necessary according to the CPs already detected. The key idea behind PAPM is that not all buffers are always needed for separate specific flows. Since deterministic routing schemes imply that a given flow from a source to a destination will be always forwarded through the same path, we can easily determine which buffers are needed for delivering such flow (those along that path). In this way, we can safely power on or off router buffers dynamically to fit the current traffic pattern requirements.

However, our proposal requires to know only those flows crossing CPs, as they are the ones using extra-queues. Each end-node, by inspecting its local memory can know the current CP locations. From that information we need to deduce the affected flows.

In addition, one key aspect of PAPM is related with the propagation of power-gating signals. Most of the current proposals for power-gating stand up for driving the modules power signals by means of dedicated wires. This strategy suffers from scaling issues in large network sizes. Instead, since PAPM needs to power buffers on/off by building

data paths according to the routing policy, it follows an strategy in which buffers are powered on in the same order a given congested message would follow through its path to destination. Taking this into account, we design PAPM to send the powering on/off signals as part of regular single flit messages sent through the regular network.

8.4.2 PAPM

Each extra-VN buffer belonging either to a router port or to a NI must be powered on only if congested traffic would potentially be delivered through that port or NI. In the case of NIs, it is quite simple since they own the ICARO cache memory in where *CPs* are stored. As soon as a NI allocates a CP (the cache is not empty), the NI becomes a potential injector of congested traffic. Therefore, its extra-VN buffer must be powered on.

Regarding router buffers at input ports, to know whether an extra-queue needs to be powered on or off becomes more difficult. Figure 8.4 shows the NIs subset (NIs 6, 7 and 8) reaching the south port of router 4 (we assume XY routing). This means that each buffer at each router must be powered on when any of those NIs has congested traffic to inject and the congested point is along those paths. In the same way, each extra-VN buffer must be powered off when none of those NIs has congested traffic to inject through those points. To manage this, when a NI stores a new *CP*, it sends an special message (*allocation message*; *AM*) to the first node reachable through the *CP* port (node 1 in Figure 8.4). This message will increase by 1 a counter stored at each input buffer at each router along the path until its destination. The extra-VN buffer is powered on when the counter is greater than zero and is powered off otherwise. In the example shown in Figure 8.4, the counter at south input port at router 4 will have a value of 3 (one for each NI).

When a given NI receives a non-congested notification, the CP is removed from the cache memory and sends a message (*deallocation message*; *DM*) to the first node reachable through the CP (node 1), causing the buffer counters along the path to be decremented by 1. In this way, when all NIs able to reach a given buffer deallocated the CP, the buffer counter will reach zero, powering off the extra-VN buffer. Note that, *AM* messages are injected through regular-VN buffers (the extra-VN buffers could be powered off), while *DM* messages are sent through the extra-VN in order to guarantee that no out of order delivery arises between the *DM* and the congested data being forwarded through the extra-buffers.

Note that, although there are works proposing to switch off portions of buffers as in [58], in this case is useless since ICARO-PAPM switches off *extra-VN* buffers, which are intended to deliver huge amounts of traffic (congested traffic). Therefore, it is expected to require the whole buffer once its required.

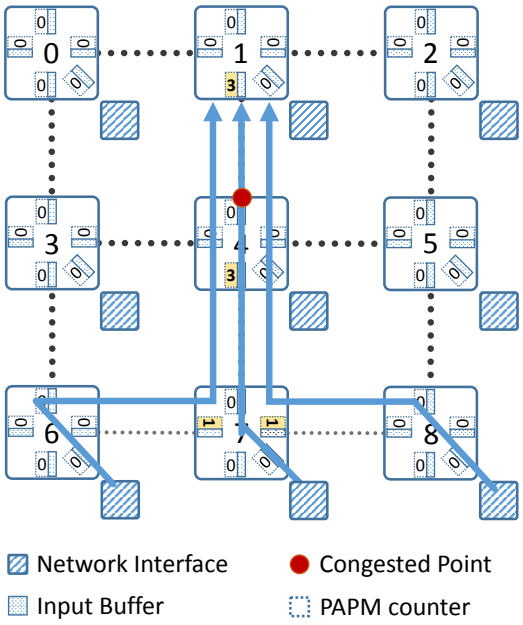


FIGURE 8.4: Network Interfaces reaching south port of router #4.

8.4.3 Selective Broadcast

Once a *CP* is detected, ICARO reallocates all traffic traversing that *CP* into the extra-VN buffers, keeping the traffic in this VN until it arrives to destination, regardless the path followed after crossing the *CP*. It means that, for a given *CP* and source, there is only one path to reach the *CP* but, beyond the *CP*, congested flows might follow different routes, therefore, all extra-VN buffers along each possible path must be powered on for the given source and *CP* pair.

To address this issue we propose a *selective broadcast* mechanism. It consists in modifying the routers behavior when forwarding *AM/DM* messages. These messages are forwarded as unicast messages until they arrive to a *CP*. Once a given *AM/DM* message crosses the *CP* (in the next router), the message is treated as a regular broadcast message. *AM* and *DM* messages are single flit messages, avoiding deadlock issues in wormhole switched networks when combined with broadcast support.

This mechanism is implemented by setting the next reachable node after the *CP* as the *AM/DM* message destination. These messages are provided with two bits: the *MCactive_bit* and the *MC_bit*. The first one is set to 0 by default and is enabled only when it arrives to the destination router (the next router reachable through the *CP* port) and the MC_bit is set to 1. When both bits are set, the message is treated as a broadcast message. Since the broadcast mechanism works by duplicating the main message, it will carry this bit to all its forked messages and this process is replicated by each forked message along its path to its destination, crossing all reachable routers starting from the *CP*. An example of an *AM/DM* message being delivered can be seen

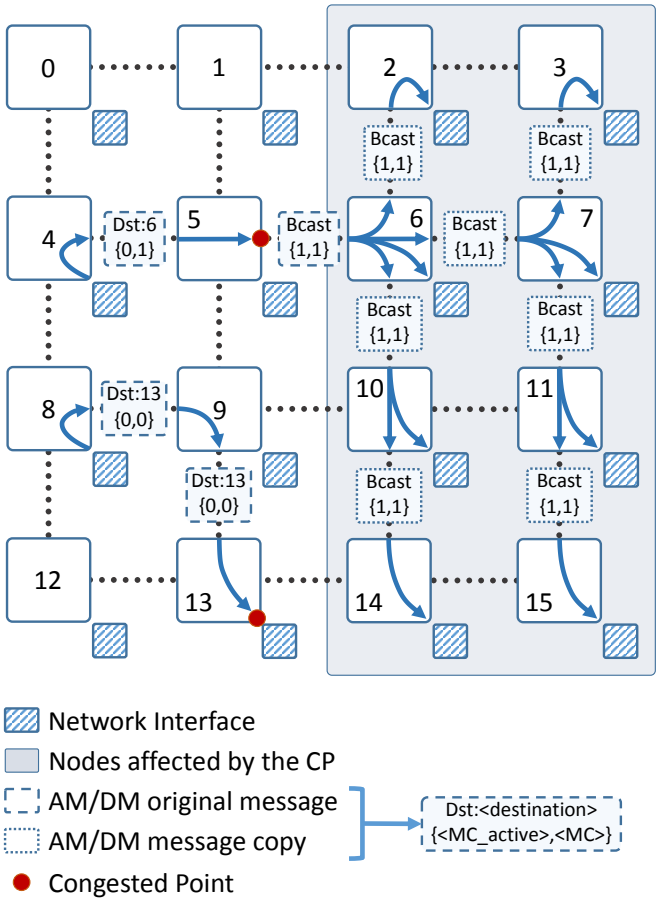


FIGURE 8.5: PAPM messages copies destinations.

in Figure 8.5. The *MC* bit, when reset, disables broadcast operation. This is useful when the CP is detected at an end-point. By forcing the *MC* bit to zero, the message is not broadcasted at the last router where the end-point is connected to. An example of the use of *MC_active* bit and *MC* bit is shown in Figure 8.5.

8.4.4 Flow Control

When using buffer power-gating mechanisms, one of the key challenges consists in avoiding race conditions. Routers are unaware of the buffers state of their neighbors, therefore, communication between adjacent routers becomes essential to know when is safe to forward data to the next buffer. For this purpose, PAPM implements a simple handshake protocol between adjacent routers. When a given buffer has to be powered off (its counter reached 0 and the buffer is empty). The router sends an *OFF_request* to the upstream router for the corresponding port. Next, when the upstream port receives the *OFF_request* sets a bit that disables the output port for being selected for delivering more flits. In addition to this, the router checks for flits having already assigned any VC for such output port. If not, an *ACK* is sent to the downstream router. If there

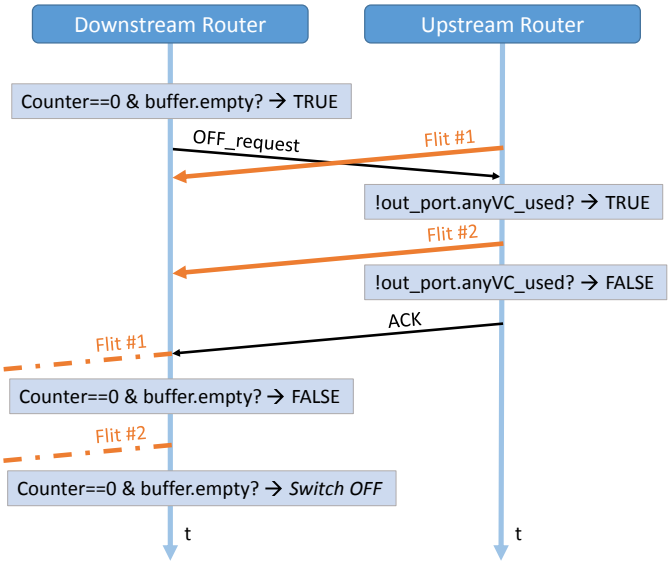


FIGURE 8.6: Buffer powering on/off protocol.

is any flit owning a VC for such output port, the VA/SA stage stops assigning VCs for that output port and the ACK is sent when no more messages own a VC for that output port. Finally, when the *ACK* is received by the downstream router, the buffer is powered off. However, it is worth to note that, between the *OFF_request* and the *ACK* arrival, some flits might arrive from the upstream router, and they must be forwarded. Therefore, actually, after receiving the *ACK*, the downstream router checks the corresponding buffer and powers the buffer off only if the buffer is empty. Otherwise, the buffer is marked as *requested to be powered off (RSO)* and, finally, the buffer is powered off when it gets empty. However, due to congestion transients, a buffer being powered off due to a recently disappeared CP, may again be requested to be powered on because de allocation of a new CP reachable also through the same path. To support this, the powering off algorithm will cancel the RSO state in case of the counter to be increased from 0 to 1 due to reception of an AM message while in RSO state. An example of this protocol is depicted in Figure 8.6.

The protocol for powering a buffer on requires only to send an *ON_request* handshake message from the downstream router. Since to power a buffer on can not generate any race condition, this message only causes the upstream buffer to enable that port to be selected in the VA/SA stage, allowing data to be forwarded to the downstream router through the extra-VN. No response from the upstream router is needed.

TABLE 8.2: General system configuration.

Network configuration	
Topology	8x8 2D mesh
Routing policy	XY
Switching technique	Wormhole (flit-level)
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4 per Virtual Network

8.5 Experimental Results

8.5.1 Methodology

In this section we report simulation results obtained for several system configurations. All simulations are performed with an in-house network-on-chip simulator under synthetic traffic patterns. Since ICARO's goal is to isolate congested traffic from non-congested ones, the inspected traffic pattern is composed of two components: uniform traffic at low data rate and hotspot traffic consisting in 4 nodes injecting to a single node. This compound synthetic traffic emulates a system in which regular data exchanged between nodes is represented by the background traffic, and main memory controller requests or hardware accelerators traffic (e.g.: cache prefetchers[79], video encoders[80]), which tend to generate hotspots naturally, is represented by the hotspot traffic. The background traffic is injected at a constant rate all simulation long. However, to emulate burstiness, the hotspot is only activated from time $300\,\mu s$ to $350\,\mu s$ (highlighted with vertical blue lines in the figures).

Regarding routers, we model a 4-stage pipeline router: IB (input buffer), RT (routing), VA/SA (virtual channel and switch allocation), X (link crossing). Since ICARO uses an additional VN to isolate congested traffic, all simulations are performed with 2 VNs, each one containing the same number of VCs. The rest of the system configuration parameters are described in Table 8.2. To elaborate different configurations, in order to evaluate our proposal, we set a *baseline* configuration. Starting from this configuration, we modify different parameters (mesh size, router's queue size, number of VCs and number of hotspots) to elaborate a set of benchmarks. All configurations are detailed in Table 8.3.

To compute power consumption we use Orion v3.0 compiled as a library for our simulator so that the simulator calls Orion periodically with the current network state to get power values of the whole mesh.

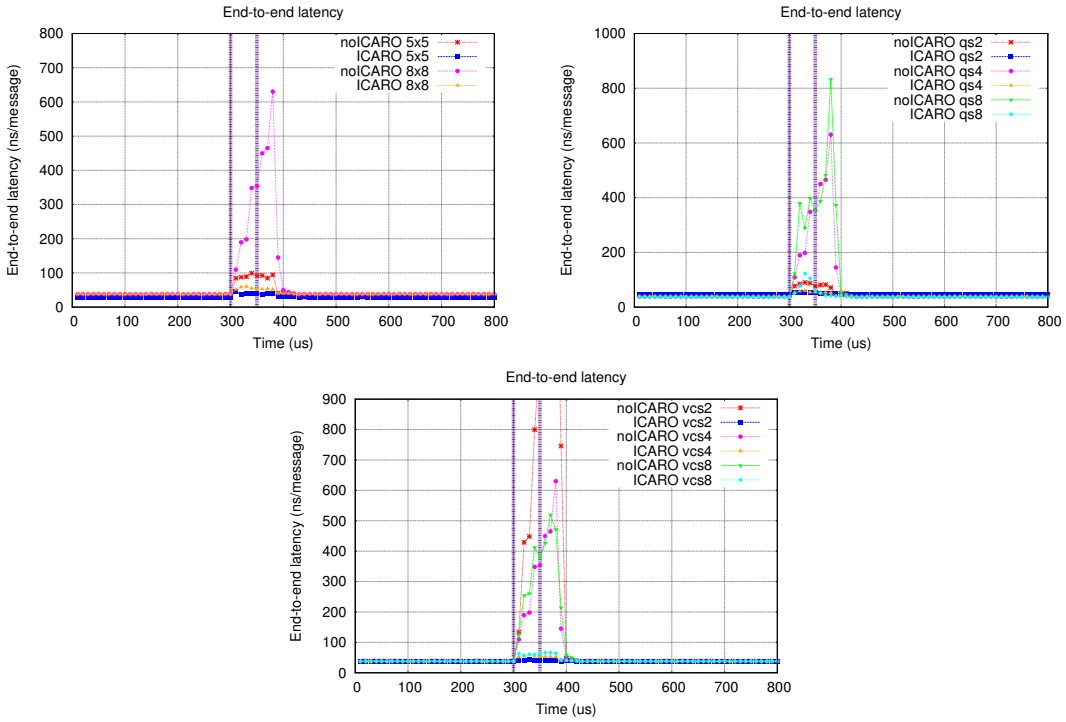


FIGURE 8.7: End-to-end latency comparison between no-ICARO and ICARO for different configuration parameters

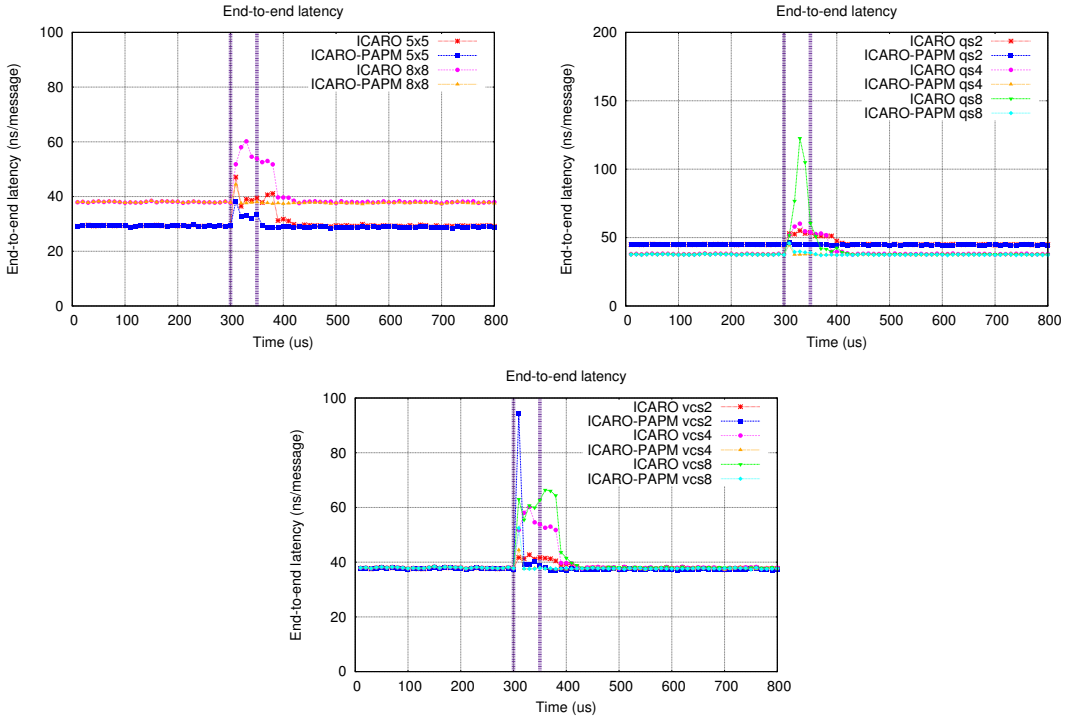


FIGURE 8.8: End-to-end latency comparison between ICARO and ICARO-PAPM for different configuration parameters

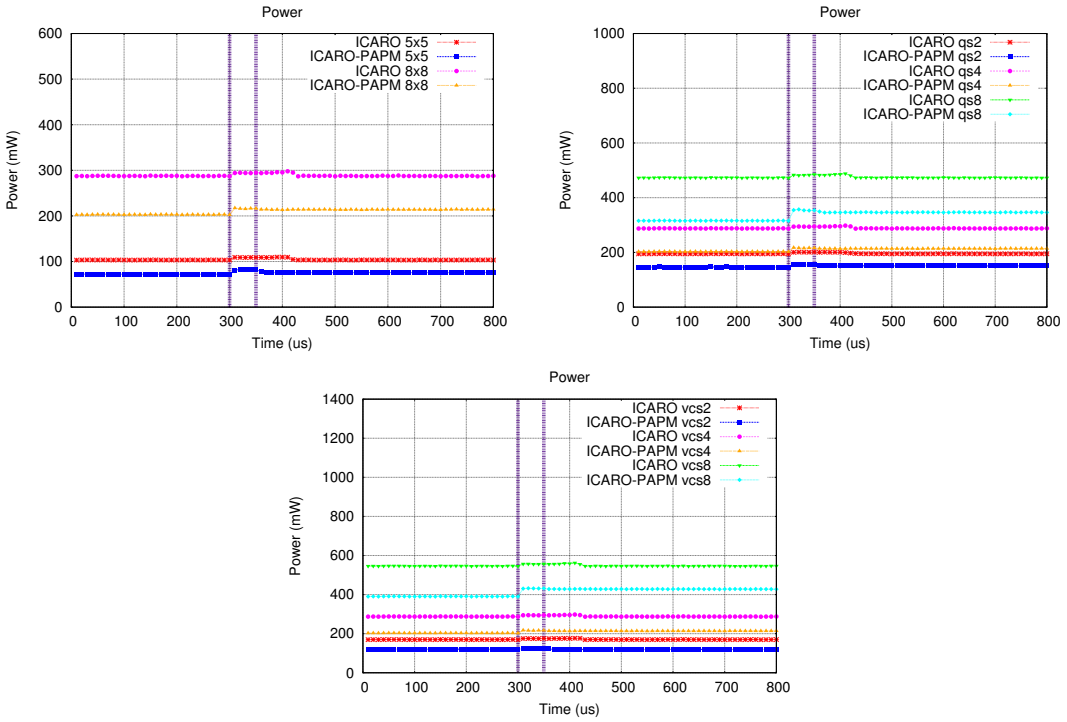


FIGURE 8.9: Power consumption for different configuration parameters

8.5.2 Results

Since the goal of ICARO-PAPM is to isolate congested traffic in order to keep the background one unaffected, for clarity, latency graphs show results only for background traffic. Note that, latencies for AM/DM messages are also included in the results. Figure 8.7 shows the latency for all configurations comparing a system without ICARO and the same scenarios with ICARO. As seen, if ICARO is not used, when the hotspot is enabled, the background traffic latency increases dramatically due to its interaction with congested flows. Nevertheless, when using ICARO, congested traffic is isolated so that background traffic keeps almost unaffected. In Figure 8.8, a comparison between ICARO and ICARO-PAPM is performed in order to demonstrate that, to implement a power-gating mechanism, has negligible effects over ICARO. As seen, ICARO-PAPM performs quite similar to ICARO. It is worth to note that, even for configurations in which PAPM messages suffer from high latencies (shown later), there is no significant ICARO-PAPM impact on performance. Regarding power consumption, in Figure 8.9 all results are depicted, showing values for the overall mesh power consumption. As can be seen, before the hotspot is activated, ICARO-PAPM saves up to 35% of power consumption by keeping the *extra-VN* buffers powered off. When hotspot is activated, ICARO-PAPM activates only those buffers that compose the routes needed to deliver congested traffic, therefore, reduces power consumption by 27%. In Figures 8.10 and Figure 8.11, final results for power consumption are shown dividing the power consumption in two sets: when congestion does not arise in the network (hotspot disabled) and when congestion

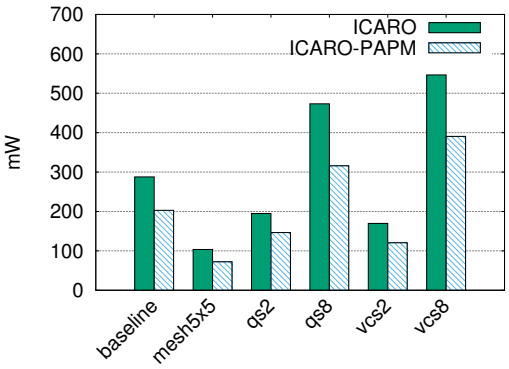


FIGURE 8.10: Average power consumption when no congestion in the network.

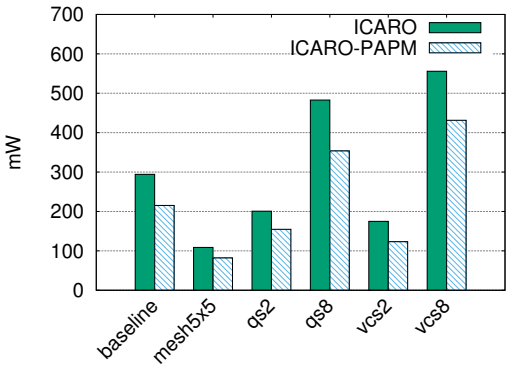


FIGURE 8.11: Average power consumption when congestion traffic in the network.

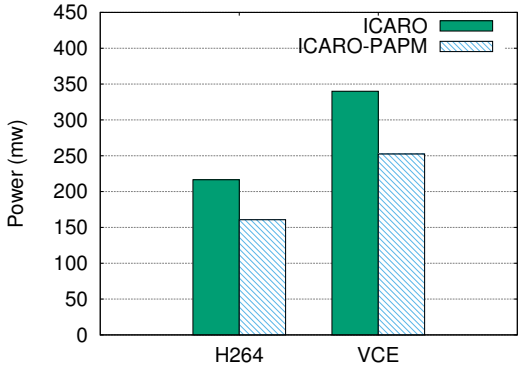


FIGURE 8.12: Average power consumption under realistic multimedia traffic patterns.

arises in the network (hotspot enabled), respectively. In all the cases ICARO-PAPM succeeds in reducing power consumption significantly.

8.5.3 Multimedia Traffic

In order to evaluate our proposal under more realistic scenarios we also performed evaluations under more realistic traffic patterns. We followed the same strategy as in [54]. We implement H264 and VCE multimedia codecs realistic traffic patterns and performed simulations with no ICARO and with ICARO-PAPM. Figure 8.12 shows the power consumption for the given traffic patterns without any congestion control and with ICARO-PAPM. As seen, ICARO-PAPM saves power by activating buffers only when necessary, achieving 26% of power saving in both scenarios with no performance loss.

TABLE 8.3: Scenarios configuration.

Scenarios				
Label	Mesh Size *(nodes)*	Queue Size	Num. VCs	Num. HS
Baseline	8x8	4	4	1
5x5	5x5	4	4	1
qs2	8x8	2	4	1
qs8	8x8	8	4	1
vcs2	8x8	4	2	1
vcs8	8x8	4	8	1

8.6 Related Work

8.6.1 Congestion Management

In congestion management most of the proposals are based on measuring congestion metrics and to use adaptive routing to skip congested nodes as in [20][91][22]. However, these approaches tend to move congestion from one region, to another. In [20], authors propose an adaptive routing policy (RCA) based on different network state metrics. RCA uses a low-bandwidth monitoring network so that, at each hop, the router aggregates its local congestion estimation based on these metrics. To aggregate this estimation, the router combines its local metrics with those received from its neighbors giving different weights to the local metrics and to the neighbors metrics. In this way, RCA, routes traffic through the best path. Weights assigned to the local and remote metrics seem critical to determine the best path. In case of assigning too much weight to the remote metrics, the local router may take wrong decisions based on the state of very far away routers. On the other hand, if too much weight is assigned to the local metrics, the routing policy may hide heavily congested nodes located far away from the router.

Other approaches opt for using also adaptive routing but based on predictions. In this sense, in [24] an algorithm based on Artificial Neural Networks (ANN) predicts the appearance of hotspots in order to react promptly. The ANN can be trained online or offline and then monitors the buffer utilization of each router to predict the formation of hotspots. However, this proposal may suffer the same issue noted previously for adaptive routing based mechanisms. In [91] authors also propose a learning-based mechanism (QCA). This mechanism uses of the Q-learning algorithm to globally and locally evaluate the network state in order to take routing decisions properly. Each time a packet is delivered from a router to its neighbor, the neighbor sends back a *learning message* containing network state information used to take routing decisions. However, this proposal requires the use of a table to store network state data on each node. This table stores data for each node, which means that the total stored data in the network grows quadratically with the number of nodes. Other authors follow other completely different strategies to tackle congestion. In [92] authors modify the router architecture

in order to allow to use other input ports buffers with free slots to store flits originally destined to other full input port buffer. This enables to optimize the use of the network resources. However, as authors admit, this leads to out of order delivery. Besides, this buffer architecture requires buffers to be provided with two writing ports, increasing the logic overhead. In addition to this, under saturation situations, congested flows could be reallocated into congestion free buffers, helping to propagate congestion along paths that, otherwise, may not be affected by congestion.

These congestion management mechanisms are focused on either dealing with congestion through adaptive routing, which is an approach that tends to move saturated regions from one network location to another or other approaches with several issues that makes them not to be affordable. Instead of this, ICARO deals with congestion from another perspective. ICARO assumes that congestion is not the problem itself, the real problem is the harmful effects (HoL) that congested flows cause to non-congested ones.Therefore, ICARO tries not to remove congestion but to isolate it.

8.6.2 Power Gating

Most of the solutions in the literature focus on applying coarse-grained power-gating techniques powering off routers or even sets of routers (regions). This is the case of [41], in which authors propose *Router Parking* (*RT*) in which routers associated to sleeping cores are powered off. They use a centralized controller (*Fabric Manager*) which collects the state of the network, takes the decisions of powering on/off each router and sends this decision to each router. Since powering routers off causes to break data paths, this proposal needs to reroute traffic around parked routers, which might increase latency and power. To deal with this, authors propose 3 different *RP* flavors: RP-A (aggressive) which parks as many routers as possible to improve power savings, RP-C (conservative) which carefully selects a small set of routers to be parked, and RP-Adp (adaptive) which selects between RP-A and RP-C dynamically depending on network utilization. This work achieves large power savings but to power whole routers off makes the complexity of this proposal to increase due to the traffic detours and the need to handle corner cases caused by network routing reconfiguration.

In [43] authors propose to power routers off but enabling bypasses at powered off routers in order to enable a guaranteed path at each router. This bypass also enables the NIs to inject and eject traffic to/from the network even when its associated router is powered off. For the routers overhead is low as it needs an inexpensive logic. However, the complexity associated to the bypass flow control and VC selection is moved to the NI, which may increase its complexity and power consumption.

Other proposals more similar to PAPM, propose to power off only the buffers, since they are the most power-hungry part of the router. For instance, in [46] a buffer power-gating

mechanism is proposed making use of *lookahead routing* to offset the amount of time necessary to powering the buffer on. By using *lookahead routing* each node is able to know in advance the path the message will follow 2 hops away from it. Each router is connected to the routers located 2 hops away on each dimension so that the n-th router is able to request the buffer powering of the *(n+2)*-th router buffers. In this way, the buffers are powered on a few cycles before the first flit arrives to the (n+2)-th router. However, this proposal requires to wire from each router to the one located at two hops on each dimension, which may be expensive.

Most of the described power-gating works are intended to switch entire routers off. However, as seen in Figure 8.1, to power off the entire router present more challenges than benefits. Also, dedicated wires to power on/off are commonly used. Our proposal, although it is designed as part of ICARO, uses a different approach, using the regular network to power on/off buffers along paths according the routing policy, which fits perfectly the needs of ICARO.

8.7 Conclusions

In this paper we have revisited ICARO, a congestion control mechanism that works by isolating congested traffic by means of an additional VN. We propose PAPM, a buffer power-gating mechanism that takes into account the paths followed by congested traffic in order to power on only the buffers needed, and added it to ICARO to alleviate the power overhead caused by the additional buffers required. We have demonstrated that ICARO behavior keeps unaffected when power-gating its additional buffers and we achieve to reduce the power consumption by up to 35% when no congestion arises on the network and up to 27% when buffers are powered on to deliver congested traffic.

Acknowledgment

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2015-66972-C05-1-R.

Chapter 9

PAPM: Path-Aware Fine-Grained Virtual Channel Power Management

- **Authors:** José Vicente Escamilla (Universitat Politècnica de València), José Flich (Universitat Politècnica de València) and Mario R. Casu (Politecnico di Torino)

9.1 Abstract

Power consumption minimization is vital in SoC designs. As technology node goes deeper, more complex systems are integrated on the same chip, driving the multicore evolution to the manycore era. The NoC plays a vital role as communication backbone and has an increasing impact on the power consumption of the whole chip. In this paper we propose PAPM, a path-aware power management strategy. PAPM allows to selectively power on and off router buffers based on the paths used by the communicating flows. In addition, PAPM detects and correctly manages shared buffers by disjoint unrelated paths. With PAPM, only buffers needed for current communication flows are powered on, thus saving large amounts of energy. In addition, complete routers may safely turn down, avoiding flow detours, when all their buffers are powered down by PAPM. Router modifications are negligible in PAPM. Results show that with PAPM we achieve up to 79% of power improvement over systems not provided with power-gating techniques and up to 45% of improvement over a recent proposed power-gating mechanism.

9.2 Introduction

Manycore processors are one of the most promising solution for addressing the never stopping search for performance improvement. Recent commercial solutions already reach hundreds of cores on the same chip (e.g. 256-cores with the MPPA architecture [78]). These systems embed an on chip network, known as a network-on-chip (NoC) [6]. The NoC is in charge of bringing effective communication between all the chip components, mainly including the cores, the memory elements and the I/O components. With every system size increment, the NoC plays a larger role in terms of performance impact. As the NoC gets larger in size, the performance of the whole system becomes affected by the performance and effectiveness of the NoC.

For a canonical NoC router, the components that consume most of the power are the buffers. Fig 9.1 shows the power consumption break down for each main component in a router. Results shown correspond to a 5-port router with 4 virtual channels (VCs) capable of storing 4 flits each. Flit width is set to 128 bits. As we can see, 70% of total power consumption is located at the input buffers.

There are many proposals in the literature for reducing power consumption in NoCs. They can be classified by the component they target. In one flavor the complete NoC router is powered down, thus maximizing power saving. This is the case of the Router Parking proposal [41]. However, in this type of strategies, routing of packets gets severely impacted, needing strategies to de-route messages when they face powered down routers.

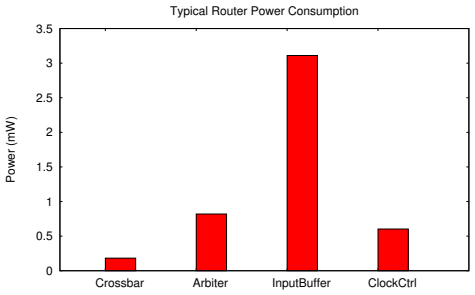


FIGURE 9.1: Power consumption of the different components of a canonical router.

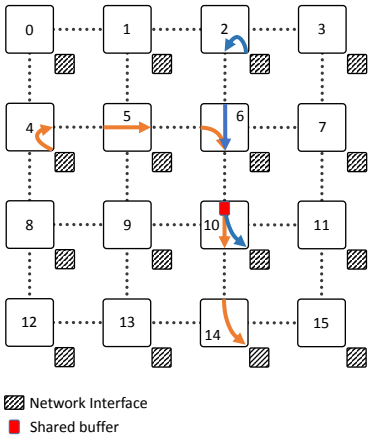


FIGURE 9.2: Flows sharing buffers along their paths.

Also, in some chip multiprocessor systems (CMPs) the last level cache is usually distributed over the complete system. Therefore, switching off a router may prevent accessing one cache bank.

Another research direction in power saving targets powering down router buffers. In this approach, the messages may encounter down buffers along their paths and two possibilities exist. The first one is to implement a mechanism to switch on buffers while the message gets temporarily blocked. The second one is to de-route messages as performed typically when switching off complete routers.

Switching off and on buffers on demand is an smart approach to save energy in the network. However, deciding when to switch on and off the buffers may become critical for performance reasons. Typically, some cycles need to be spent in the process of powering on a buffer. During those cycles messages get blocked potentially incurring also on temporal congestion in the network. Also, the process of powering on and off buffers incurs in energy overhead. Indeed, the number of cycles a buffer must be switched off in order to amortize this overhead is known as *Break Even Time (BET).* Thus, it is important to switch on or off buffers appropriately to avoid wasting power.

On the other hand, NoC routers typically implement different buffer queues at the input ports in order to provide higher performance. Indeed, when using virtual channels, messages blocked temporarily may get bypassed by other messages, thus improving performance. Virtual channels can be used for other purposes, for instance deadlock avoidance and for implementing different virtual networks on the same physical network. Anyway, from a power consumption perspective, the use of virtual channels incurs in more power overheads due to the need of using more buffers.

The effectiveness of using virtual channels for performance improvement depends on the load of the network. If the network load is low, more than one queue per input

port (one virtual channel) will not be needed most of the time. Indeed, messages will seldom block in a low loaded network. However, when the load of the network increase, more contention will occur and messages will interfere between them. In that sense, having more than one virtual channel will help in reducing conflicts and achieving good performance.

In this scenario (NoC with virtual channels and a varying network injection rate), power consumption management becomes complex. Moreover, when running several concurrent applications, most router buffers will be shared by all the applications. It would be advisable to have an smart NoC which effectively knows which parts of the network (indeed the buffers) are needed by which applications, and therefore, switching on only those buffers when needed.

In this paper we address this challenge and propose PAPM (Path-Aware Power Management) strategy. We propose an strategy and describe an implementation for the strategy. PAPM relies on the paths used by the application in order to manage the buffers along those paths. PAPM will control also buffers at routers shared by different applications (or by different flows of the same application), thus preventing those buffers being powered down even if one application finishes its communication process. Moreover, routers with all the buffers powered down will be powered off completely thus saving more energy.

Results show that our proposal achieves up to 80% of power consumption improvement over a system provided with no power-gating mechanism. Also, we overcome by up to 57% a recent state-of-the-art proposal.

The rest of the paper is structured as follows. First, we describe the state-of-the-art related to power-gating in NoCs. Then, we describe PAPM. Next, we evaluate our proposal and analyze the results. Finally, we present the conclusions of this work and our future plans.

9.3 Related Work

There are many proposals for power-gating. Traditionally, most of the proposals intended to networks-on-chip consist in powering-off routers completely (coarse-grained power-gating). However, in order to disable a router completely, all traffic traversing the router must be diverted through alternative paths, which may have a significant impact in performance. Among these proposals we can highlight Router Parking[41]. This work takes advantage of sleeping nodes to disable their routers in order to save power. To do this, a centralized manager (FM) is used, which is in charge of collecting the state of all nodes and to drive the network reconfiguration. Reconfiguration is performed in periods of several μs. This prevents taking advantage of more potential opportunities

within this lapse of time. Additionally, network reconfigurations may take also several μs, which may cause penalties from the point of view of performance. Authors note that a centralized manager performs better than a distributed mechanism. However, it is well known that centralized strategies typically do not scale for large systems and become single point of failure for the system.

Disabling paths by switching buffers off can compromise performance, specially when using deterministic routing. To solve this issue, some works, instead of switching all the buffers off in a given port, decrease the available resources (number of VCs) by switching some of the buffers off[42] or switch all buffers off and enable *bypasses* to preserve paths. In [43] authors propose to power routers off but enabling bypasses at powered off routers in order to enable a guaranteed path at each router. This bypass also enables the NIs to inject and eject traffic to/from the network even when its associated router is powered off. For the routers, the overhead is low as it needs an inexpensive logic. However, the complexity associated to the bypass flow control and VC selection is moved to the NI, which may increase its complexity and power consumption. In [44] authors propose TooT, which relies on the fact that most of the traffic crosses routers making no turns. Based on this fact, TooT switches most parts of the router off, and keeps on only a very reduced version of the router and one latch per port, allowing to forward traffic that requires no turns. However, effectiveness of this sort of strategies depend on the traffic pattern and may not work properly under some circumstances.

Other works advocate for using fine-grained power-gating, dividing routers into small parts or domains capable of being switched off individually. This is the case of *Power Punch*[47]. This proposal consists in sending power-gating signals through the network in order to switch buffers on/off as long as these signals are delivered through buffers. The key of this work is the way in which the authors propose to aggregate in the same message different power-gating signals during their delivery through the network, alleviating the overhead caused by these signals.

Since one of the key challenges of implementing power-gating techniques is to deal with the delays derived from the buffer power switching some authors take benefit of look-ahead routing. Related to this, in [46] a buffer power-gating mechanism is proposed using *lookahead routing* to offset the amount of time necessary to powering buffers on. By using *lookahead routing* each node is able to know in advance the path the message will follow 2 hops away from it. Each router is connected to the routers located 2 hops away on each dimension so that the n-th router is able to request the buffer powering of the *(n+2)*-th router buffers. In this way, the buffers are powered on a few cycles before the first flit arrives to the (n+2)-th router. However, this proposal requires to wire from each router to the one located at two hops on each dimension, which may be expensive.

Our proposal overcomes most of the drawbacks of other works. PAPM is aware of the actual network requirements by keeping track of required paths to serve flows before

injecting at sources allowing to enable resource very accurately and avoiding traffic detouring. This information is used to fastly deliver power-on signals through a lightweight network avoiding to saturate the regular network with several messages. Also, our proposal works at a fine-grained level, first driving the buffers and finally over the rest of the router logic, which increase opportunities of saving power.

9.4 PAPM Description

In this section we describe the PAPM strategy and a possible implementation. First, we provide a general description of the strategy and then we focus into implementation details.

9.4.1 General Description

The PAPM method works at the granularity of paths. One path is defined by the source and destination end-nodes connected through the NoC. Those nodes use a fixed path (we assume deterministic routing) to communicate. Along this path, a set of buffers are used to flow control the advance of the traffic. Therefore, a path can be seen as a chain of buffers.

PAPM manages the status of all the buffers along a path. Buffers can be powered on or off. Whenever a path needs to be used, the source end node injects a control message, referred to as ABP (Activate Buffers Path), in order to power on all the buffers along the path. Those buffers will then be kept on during the transmission of traffic along the path. When the source node has no more traffic to inject or when it decides to temporarily switch off the path (to save energy) then the node injects a similar message, referred to as DBP (Deactivate Buffers Path), in order to power down buffers along the path.

One important aspect of PAPM is to be fast enough when powering on buffers. Indeed, those buffers need to be on for the transmission of incoming messages. To speed up this process, the PAPM method will rely on a lightweight fast network implemented as a bidirectional ring. This network, referred to as *Activation Network (AN)* will enable powering on all the buffers along a path.

The power down process in PAPM (DBP message) works, however, in synchrony with the transmission of messages. Indeed, PAPM will inject the DBP message through the regular NoC network, potentially switching off buffers along its way to final destination.

One key aspect of PAPM is the management of shared buffers by concurrent flows. Indeed, two non-disjoint paths in the network will share some input ports, and thus,

buffers. Activating and deactivating buffers for one path does not have to conflict with the expected buffer status of the other path. Indeed, buffers need to be powered on if any of the paths sharing the buffer are active. Figure 9.2 shows the case.

To address the sharing buffers issue, PAPM will rely on an internal counter strategy on every router input port. The counter will increase by one for every ABP message received addressing that buffer. Accordingly, the counter will decrease by two (explained in Section 9.4.3) for every DBP message received through the associated input port. The input port will be activated (powered on) based on the counter value.

When a node allocates a message for a given destination, the NI PAPM module checks the destination in order to know whether this path is available (switched on) or not.[1] This information is stored in the *Active Paths Bitmap* by means of a bit for each destination node (1=path active, 0=path inactive). If the path is currently active, no additional action is required to send the message. Otherwise, an ABP is sent through the AN network in order to switch on the path, being marked as *active* in the paths bitmap. Note that the path may actually be switched on because other node has asked the same path to be switched on. Once the message is sent, PAPM checks whether the path must be switched off again. If so, PAPM sends a DBP message through the regular network in order to switch the path off and the path is marked as *inactive*.

In addition to switch buffers off, to save more power, PAPM monitors at each router the state of all buffers. When all buffers are off, since the rest of the router logic is no longer needed, the complete router is switched off as well. Accordingly, when any of the buffers are switched on, the router logic is also switched on.

9.4.2 Router Implementation

Figure 9.3 shows the implementation of PAPM strategy on the baseline router assumed. The router implements the standard logic blocks, namely input buffer, routing unit, virtual channel allocator, switch allocator and crossbar. The PAPM strategy impacts mainly on the input buffer strategy. A counter and a logic block is added to update the counter based on the arrival of ABP and DBP messages. Notice that DBP messages arrive through the input port associated to the buffer whereas ABP messages arrive through a lightweight new input port for the router. The added control network will deliver ABP messages through that port.

The logic to support PAPM on the router design is small. Notice that every input port needs to compute whether the port is along the path set between the source and the destination of the ABP message. This logic is provided with the notification source, the

[1] Note that there is no way to certainly know whether the whole path is effectively on/off since other nodes may share parts of the path. A given node is only able to certainly know whether it has requested to switch the path on or off.

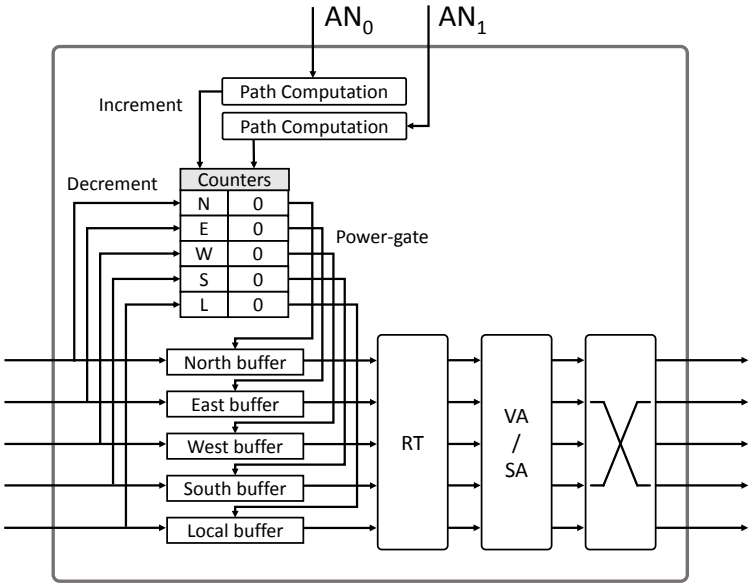


FIGURE 9.3: Router implementation.

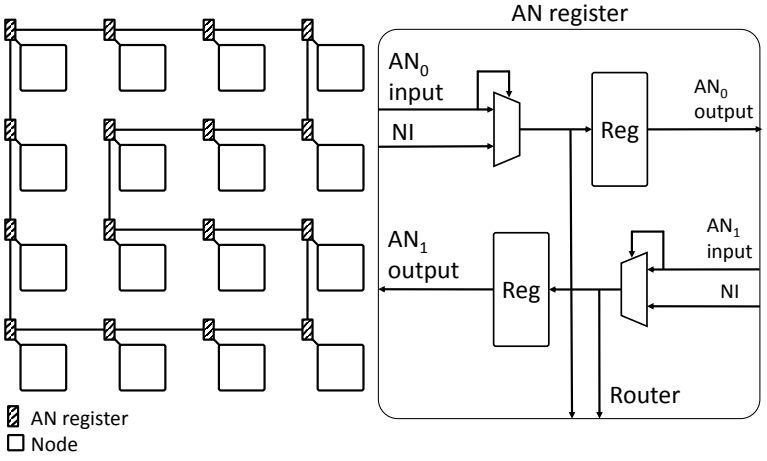


FIGURE 9.4: AN network in a 4x4 mesh.

destination node which defines the path (source→destination) to be switched on/off and the current node which allows to know whether this node belongs to the path and which ports are involved in it. The outputs of this logic are connected to the counter of each input port, allowing the counter for each buffer to be incremented when needed.

As seen in Figure 9.3, each counter for each port drives the power for each input port. These control signals trigger the actions to power on or off those related buffers.

Switching ports off may cause race conditions since neighbor routers may already have sent flits to the power-gated input port or, at least, already reserved resources to deliver flits (credits). To avoid these synchronization issues we implemented a simple handshaking protocol. Given router A wants to power one of its input ports off connected to neighbor B, this protocol simply sends a message from A to B just before switching

the port off. Router B replies to this message with an ACK signal in case there are no flits pending to be delivered to A and marks the output port as *disabled* to avoid to be selected in the VA/SA in further arbitrations. Otherwise, router B waits until the condition meets to send this ACK. Once the ACK signal is received by router A, the input port is switched off. To switch a port on, a signal is sent to router B to force router B to mark the output port as *enabled*.

9.4.3 Activation Network

Figure 9.4 shows the added control network to deliver ABP messages. The network forms a bidirectional ring topology forming a zig-zag structure visiting all the NoC routers and end nodes. For each network hop one simple latch is used and an small multiplexer unit is added. The multiplexer is added on every end-node in order to allow to inject ABP messages. The output of the demultiplexer is wired on every router in order to eject a copy of traveling ABP messages through the new network.

The ABP messages injected through the network will travel along all the ring and will be removed when they reach again the injector end node (one complete cycle performed). To do this, each message includes the following fields: source of the path (*src*), destination of the path (*dst*) and a valid bit (*valid*). In order to speedup the ABPs delivery, the ring works at twice the system clock frequency by using latches activated either by rising or falling edge. Because of this, each ABP is duplicated and injected in each direction of the ring. This means that each router will receive each ABP duplicated, thus its affected buffer counters will be increased by 2. To solve this, each DBP received will cause the buffer counter to be decreased by 2 also. Whenever an ABP message is within the network, the message will get maximum priority to move forward along the ring.

ABP messages are triggered by messages allocation and this may occur up to once per cycle and ABP injection may be blocked due to the highest priority of the in-flight notifications. Because of this, an small buffer is needed for ABPs storing. However, we analyzed empirically the required buffer size and we arrived to the conclusion that a 2-slots length buffer is enough to avoid any issue for all simulations performed in Section 9.5.

9.4.4 Power-Down Strategy at End Nodes

Switching off/on buffers incur in power penalties, potentially ruining any power saving achieved by switching them off. In order to amortize this power overhead, the buffer must be powered off a minimum number of cycles (BET). Therefore, one important aspect of PAPM method is deciding when to inject ABP or DBP messages. This is performed at the end-nodes. To do this, PAPM keeps track of the time between generation of

messages for the each destination (TBG_{dst}). At each generated message, the TBG_{dst} value is updated with the following formula:

$$TBG_{dst} = (TBG_{dst} \times 0.8) + ((T_{current} - T_{last}) \times 0.2)$$

where $T_{current}$ represents the current time and T_{last} represents the time where the last message to the same destination was injected.

PAPM implements a bit vector to keep track the status for all the paths referred to as (*Active Paths Bitmap*). When a message is generated, PAPM checks the status for the path used to reach the destination. If the path is *off*, then an ABP message is injected, changing the state of the bit to *on*.

The message generated is then delivered to the network interface queue and prepared for injection. When the tail of the message reaches the head of the queue (just before injecting it), PAPM checks whether the path has to be powered down or not. The path should be powered off if the time for the next injected message to the same destination is larger than the *BET*. This means some power saving will be achieved. Therefore, PAPM enables the DBP bit (which converts the message into a DBP message) in the tail flit of the message if the following condition applies:

$$TBG_{dst} > BET$$

If the expected time between injections is smaller then the path is not switched off.

If the last message sent to a given node did not trigger the DBP bit and no more messages are allocated for that destination node (TBG failed predicting the next allocation time) will cause the path to be kept on indefinitely. To avoid this, a dedicated module (TBG watchdog) is in charge of automatically sending a dedicated DBP message to such destination node after $3 * TBG$ cycles in case of no new message allocation.

9.5 Performance Evaluation

9.5.1 Simulation Testbed

In this section, PAPM is evaluated under different configurations using a cycle-accurate in-house network-on-chip simulator. The simulator implements 4-stage routers: IB (Input Buffer), RT (Routing), VA/SA (Virtual Channel and Switch Allocation), X (Crossbar). See Table 9.1 for further configuration details. To obtain our power results we used a modified version of Orion v3.0 [57] and Encounter tool from Cadence for calculating power overhead due to the additional logic added by PAPM.

Simulation configuration	
Topology	4x4 2D mesh
Routing policy	XY
Switching technique	Wormhole
Flow control	credits
Flit size	128 bits
Message size	10 flits
Switch queue size	4 flits
Virtual Channels	4
Frequency	1GHz

TABLE 9.1: Simulation configuration.

Time (μs)	Pattern	Inj. Rate (f/c)
0-99	uniform	0.1
100-199	bit-reversal	
200-299	bit-complement	
300-399	bit-rotation	
400-499	bit-shuffle	
500-599	transpose	
600-699	tornado	
700-799	butterfly	

TABLE 9.2: Traffic patterns.

Regarding power-on delay, according to the current state-of-the-art [45] we could assume a delay of *0.2ns*. However, in order to show the behavior of our proposal under worse cases, for our experiments we assume a power-on delay of *2ns* (2 cycles).

In order to evaluate our proposal, we perform two analysis following different approaches. First, in Section 9.5.2 we simulate a system performing changes of context by changing between different traffic patterns for two different mesh sizes: 4x4 and 8x8. Then, in Section 9.5.3, we use realistic traffic [54], increasing its intensity until network saturation to find the upper limit in which buffers are used at their maximum rate while using more realistic traffic.

For our analysis we show results for a system provided with no power-gating mechanism, results for a system implementing PAPM and, additionally we compare also with TooT[44], a recent power-gating proposal described in Section 9.3, which essentially switches routers off and bypasses ports when traffic with no turns is detected.

9.5.2 Performance Analysis

For our first benchmark we perform simulations using several synthetic traffic patterns. Simulation toggles the traffic pattern used along the time in order to emulate a system running different applications. By doing this, we demonstrate that PAPM is able to dynamically adapt the available buffers in the network to fit the application requirements while saving power switching off those which are not used. Details about the traffic patterns used are described in Table 9.2.

One of the main challenges when implementing power-gating based strategies is to hide the delay caused by the powering on process. In addition to this, since our proposal is based in notifications (ABPs) to trigger this process, there is an additional delay caused by ABPs delivery time. However, as seen in Fig. 9.5, the AN is able to deliver all notifications in time thus no significant latency overhead can be appreciated. In the same way, as seen in Fig. 9.7, no impact on throughput can be appreciated. However, as shown in Fig. 9.6, PAPM disables buffers not required to deliver each traffic type, achieving up to 73% and 33% of power savings compared with no power-gating and

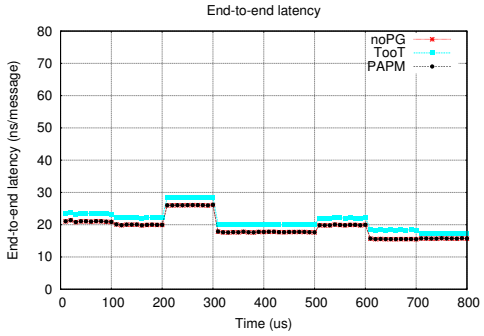


FIGURE 9.5: End-to-end latency.

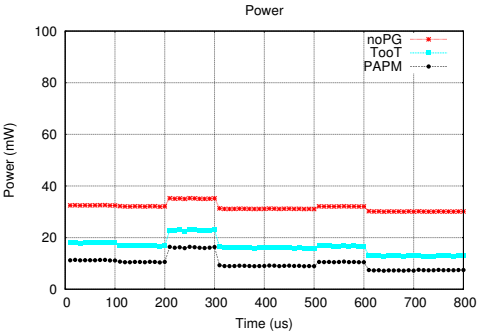


FIGURE 9.6: Power consumption.

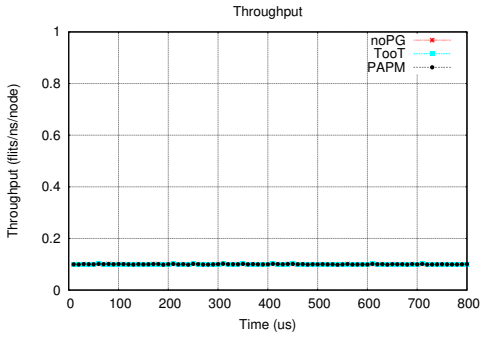


FIGURE 9.7: Throughput.

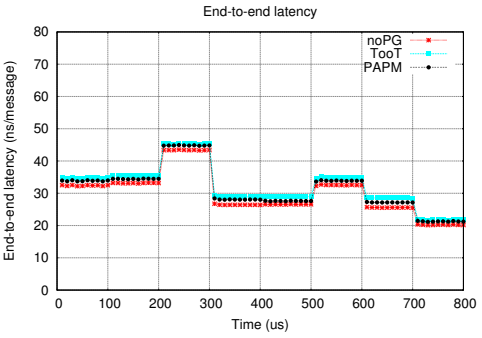


FIGURE 9.8: 8x8 end-to-end latency.

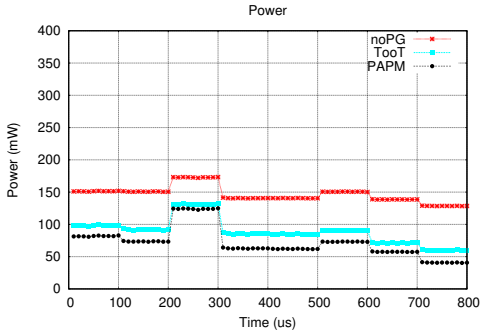


FIGURE 9.9: 8x8 power consumption.

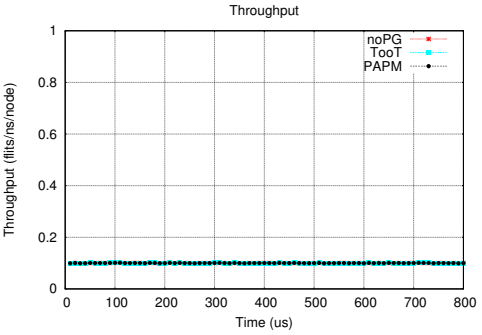


FIGURE 9.10: 8x8 mesh throughput.

TooT respectively. It is worth to note that, in addition to overcome TooT saving power, TooT suffers latency overheads while PAPM keeps the latencies unaffected. Similarly, Figures 9.8, 9.9 and 9.10 show the results for a 8x8 mesh network. As shown, the latency when running PAPM increase slightly. This is due to the delay of the AN delivering the ABPs to all nodes in the network. However, this increase is minimal, is lesser than the latency increase using TooT and the power saving of PAPM is still around 65% of improvement compared with the no power-gating case and 21% comparing with TooT.

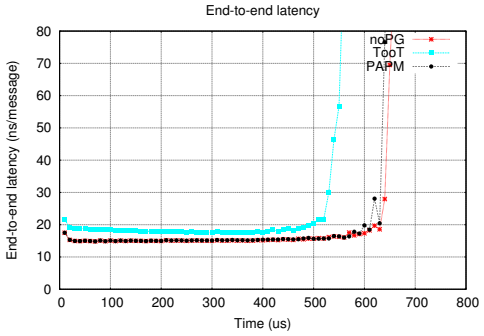


FIGURE 9.11: H264 end-to-end latency.

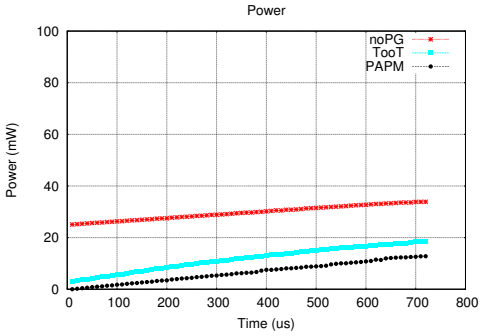


FIGURE 9.12: H264 power consumption.

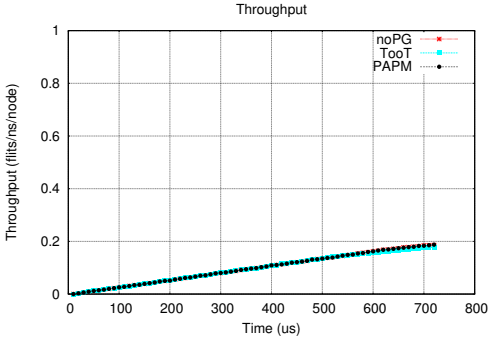


FIGURE 9.13: H264 Throughput.

9.5.3 Saturation Analysis

For this analysis a realistic traffic pattern corresponding to the H264 codec is used. To generate this traffic we followed the methodology described in [54]. Since this traffic emulates the H264 video codec traffic, we are able to provide the frame rate at which the codec works. Since flows from any source are sent always to the same destination, the subset of used buffers is always the same. Taking benefit of this, for this analysis we increase the frame rate parameter in order to increase the network load under this traffic in order to analyze how PAPM and TooT react to an increasing network load until saturation using always the same buffers subset. In addition to this, performing this analysis is also useful to evaluate PAPM and TooT under more realistic traffic.

As seen in Fig. 9.11 and Fig 9.13, PAPM has negligible effect over the latency and throughput respectively. As seen, at very low frame rate, the network utilization is very low, therefore, while the no power-gating case keeps all the buffers on, PAPM only switch them on when necessary, consuming very low power. Regarding TooT, due to the reduced buffers capacity in routers when bypasses are enabled, cause the saturation to arise earlier, which makes it not suitable under high network load. As long as the frame rate is increased, power consumption for the no power-gating case increases due to the dynamic power component and PAPM reacts by increasing the buffers uptime, increasing

the power consumption as well. However, in the worst case (at the highest frame rate), PAPM achieves an average power consumption improvement of 79% comparing with case with no power-gating and of 45% comparing with TooT.

9.6 Conclusions

In this work we presented PAPM, a fine-grained and path-aware buffer power-gating mechanism. PAPM essentially works by sending power-gating messages from each source in order to increase counters implemented in each router buffer. In this way, buffers are provided with a mechanism to determine whether each buffer is in use by any node and switching on or off the buffer adequately. We demonstrated PAPM is able to save power by up to 79% compared with a system provided with no power-gating mechanism and improves TooT by up to 45% with no significant latency or through penalty.

9.7 Future Work

As future work we plan to improve PAPM by providing it the ability to switch VCs on or off depending on the requirements of the system, offering more resources in those network points where more traffic coincides.

Chapter 10

Conclusions

10.1 Contributions

Networks-on-chip are emerging as the key solution in the multicore/manycore era to provide connectivity between tens, hundreds or even thousand of nodes, due to its effectivity and scalability. However, as the scaling technology goes further heading to the manycore era, network saturation and power consumption becomes an imminent challenge that must be addressed to guarantee next generation chips performance. Current power control mechanisms incur in performance penalties, which lead to system performance degradation. Because of this, in this thesis we have addressed the challenge by designing an effective congestion control mechanism (ICARO) based on HoL-blocking removal through congestion isolation, and then we combined this congestion control mechanism with DVFS-based and power-gating techniques.

First, we developed BAHIA to deal with bursty traffic by means of monitoring end nodes. BAHIA is able to detect bursty traffic at end nodes and effectively isolate bursts into the *extra-VN*, improving the system performance. However, bursty traffic may not necessarily be harmful to the rest of traffic since it may not cause congestion, therefore, the BAHIA detection strategy is naive in that sense. Because of this, BAHIA can be seen as a first step before tackling a more sophisticated mechanism, which is ICARO, a congestion control mechanism which works by detecting congestion at routers, which is a more complex but also more accurate and effective approach.

ICARO, follows the same traffic isolation approach of BAHIA but improves the detection strategy, effectively identifying harmful traffic and succeeds in isolating it, thereby improving the system performance due to the accurate and fast HoL-blocking removal.

Next, we combined ICARO with DVFS in three flavors depending on the parameter to be improved: performance, power consumption and a balance of both. With this work, we demonstrated that congestion isolation by means of ICARO can perform successfully in improving different parameter in systems provided with DVFS.

Also, we combined ICARO with DMSD, a latency driven DVFS-based proposal. DMSD works by setting a latency target close to the saturation point to maximize power-saving. However, irregular traffic patterns as hotspot may affect negatively to the DMSD behavior. Therefore, we combined ICARO with DMSD in order to discriminate hotspot traffic type from non-hotspot one, thereby achieving best power-savings and guaranteeing latencies for non-hotspot traffic.

As a final contribution, we proposed PAPM, a path-aware power-gating mechanism. Different from most of the work in the literature, which work at flit granularity, PAPM works at data flow granularity. PAPM works by identifying data flows and activating all buffers composing the required path in a row. Although PAPM was initially designed to improve the power overhead of ICARO due to its additional required resources, due

to its success, we also proposed a standalone version to work in regular systems without ICARO, demonstrating to overcome TooT, a recent state-of-the art power-gating proposal.

10.2 Future Directions

Two of the main challenges of power-gating consists, on one hand, not to violate the BET in order to avoid to waste power, thus compromising the usefulness of the mechanism and, on the other hand, to deliver power-gating signals as fast as possible to the component to be woken-up. Therefore, our future plans consist in following the approach of the standalone version of PAPM but improving the paths request detection by means of traffic predictors based on the locality principle and, in order to make power-gating signals to be delivered faster, we plan to implement hybrid or clustered networks, thereby improving its scalability and speeding up power-gating signals delivery.

10.3 Publications

Conferences:

- J. V. Escamilla, J. Flich and P. J. Garcia. Burst-Aware HoL Blocking Avoidance. In *Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems* (ACACES), pages 237-240, Fiuggi, Italy, 2012.
- J. V. Escamilla, J. Flich and P. J. Garcia, "Head-of-Line Blocking Avoidance in Networks-on-Chip," 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, 2013, pp. 796-805. doi: 10.1109/IPDPSW.2013.214
- J. V. Escamilla, J. Flich and P. J. Garca, "ICARO: Congestion isolation in networks-on-chip," 2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS), Ferrara, 2014, pp. 159-166. doi: 10.1109/NOCS.2014.7008775
- Escamilla J.V., Flich J., Garca P.J. (2015) Efficient DVFS Operation in NoCs Through a Proper Congestion Management Strategy. In: Hunold S. et al. (eds) Euro-Par 2015: Parallel Processing Workshops. Euro-Par 2015. Lecture Notes in Computer Science, vol 9523. Springer, Cham
- J. V. Escamilla, M. R. Casu and J. Flich, "Increasing the Efficiency of Latency-Driven DVFS with a Smart NoC Congestion Management Strategy," 2016 IEEE

10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), Lyon, 2016, pp. 241-248. doi: 10.1109/MCSoC.2016.42

- J. V. Escamilla, J. Flich and M. R. Casu, "ICARO-PAPM: Congestion Management with Selective Queue Power-Gating,"2017 IEEE 15th International Conference on High Performance Computing & Simulation (HPCS), Genoa, 2017

In addition, other related papers have been published in national conferences:

- J. V. Escamilla, J. Flich and P. J. Garcia. BAHIA: Burst-Aware Head-of-Line Blocking Injection Avoidance. In *Actas de las XXIII Jornadas de Paralelismo* (JP), pages 351-356, Elx, Spain, 2012.
- J. V. Escamilla, J. Flich and P. J. Garcia. Congestion Isolation in Networks-on-chip. In *Actas de las XXIV Jornadas de Paralelismo* (JP), pages 24-29, Madrid, Spain, 2013.
- J. V. Escamilla, J. Flich and P. J. Garcia. Mejorando DVFS en redes en chip mediante tcnicas de control de congestin. In *Actas de las XXVI Jornadas de Paralelismo* (JP), pages 250-257, Cordoba, Spain, 2015.
- J. V. Escamilla, Mario R. Casu and J. Flich. Guaranteeing latencies in DVFS-based NoCs under unbalanced traffic loads. In *Actas de las XXVII Jornadas de Paralelismo* (JP), pages 431-439, Salamanca, Spain, 2016.

References

[1] Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, M. Pedram, and Naehyuck Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(5):695–708, May 2013. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2235126.

[2] Wikipedia. Intel 80286. Available at `https://es.wikipedia.org/wiki/Intel_80286`.

[3] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey. A voltage reduction technique for digital systems. In *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pages 238–239, Feb 1990. doi: 10.1109/ISSCC.1990.110213.

[4] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T.N. Vijaykumar. Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*, pages 90–95, 2000. doi: 10.1109/LPE.2000.155259.

[5] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pages 684–689, 2001. doi: 10.1109/DAC.2001.156225.

[6] José Flich and Davide Bertozzi. *Designing Network On-Chip Architectures in the Nanoscale Era*. Chapman & Hall/CRC, 2010. ISBN 1439837104, 9781439837108.

[7] Tilera Corp. Tilera tile multicore processors. Available at `http://www.mellanox.com/page/products_dyn?product_family=238&mtag=tile_gx72`, .

[8] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *Solid-State Circuits, IEEE Journal of*, 43(1):29–41, Jan 2008. ISSN 0018-9200. doi: 10.1109/JSSC.2007.910957.

[9] Guihai Yan, Yingmin Li, Yinhe Han, Xiaowei Li, Minyi Guo, and Xiaoyao Liang. Agileregulator: A hybrid voltage regulator scheme redeeming dark silicon for power efficiency in a multicore architecture. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1–12, Feb 2012. doi: 10.1109/HPCA.2012.6169034.

[10] A. Scherrer, A. Fraboulet, and T. Risset. Automatic phase detection for stochastic on-chip traffic generation. In *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, pages 88–93, Oct 2006. doi: 10.1145/1176254.1176277.

[11] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *Micro, IEEE*, 23(6):84–93, Nov 2003. ISSN 0272-1732. doi: 10.1109/MM.2003.1261391.

[12] R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, Nov 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.379.

[13] AMD. What is heterogeneous system architecture (hsa)? Available at `http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-system-architecture-hsa`.

[14] M.J. Karol, M.G. Hluchyj, and S.P. Morgan. Input versus output queueing on a space-division packet switch. *Communications, IEEE Transactions on*, 35(12): 1347–1356, Dec 1987. ISSN 0090-6778. doi: 10.1109/TCOM.1987.1096719.

[15] P.J. Garcia, F.J. Quiles, J. Flich, J. Duato, I. Johnson, and F. Naven. Efficient, scalable congestion management for interconnection networks. *Micro, IEEE*, 26(5): 52 –66, sept.-oct. 2006. ISSN 0272-1732. doi: 10.1109/MM.2006.88.

[16] George Nychis, Chris Fallin, Thomas Moscibroda, S. Seshan, and O. Mutlu. Congestion control for scalability in bufferless on-chip networks. Technical report, 2011. URL `http://c1f.net/pubs/tr-2011-003-cc-scale.pdf`.

[17] M. Thottethodi, A. R. Lebeck, and S. S. Mukherjee. Self-tuned congestion control for multiprocessor networks. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 107–118, 2001. doi: 10.1109/HPCA.2001.903256.

[18] K. K. W. Chang, R. Ausavarungnirun, C. Fallin, and O. Mutlu. Hat: Heterogeneous adaptive throttling for on-chip networks. In *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, pages 9–18, Oct 2012. doi: 10.1109/SBAC-PAD.2012.44.

[19] U.Y. Ogras and R. Marculescu. Prediction-based flow control for network-on-chip traffic. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 839–844, 2006. doi: 10.1109/DAC.2006.229272.

[20] Paul Gratz, Boris Grot, and Stephen W Keckler. Regional congestion awareness for load balance in networks-on-chip. In *Proc. HPCA*, pages 203–214, 2008.

[21] M. Ramakrishna, P.V. Gratz, and A. Sprintson. Gca: Global congestion awareness for load balance in networks-on-chip. In *Networks on Chip (NoCS), 2013 Seventh IEEE/ACM International Symposium on*, pages 1–8, April 2013. doi: 10.1109/NoCS.2013.6558405.

[22] X. Chang, M. Ebrahimi, M. Daneshtalab, T. Westerlund, and J. Plosila. Pars x2014; an efficient congestion-aware routing method for networks-on-chip. In *The 16th CSI International Symposium on Computer Architecture and Digital Systems (CADS 2012)*, pages 166–171, May 2012. doi: 10.1109/CADS.2012.6316439.

[23] A Kumar, Li-Shiuan Peh, and N.K. Jha. Token flow control. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 342–353, Nov 2008. doi: 10.1109/MICRO.2008.4771803.

[24] E. Kakoulli, V. Soteriou, and T. Theocharides. Hpra: A pro-active hotspot-preventive high-performance routing algorithm for networks-on-chips. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 249–255, Sept 2012. doi: 10.1109/ICCD.2012.6378648.

[25] Yi Xu, Bo Zhao, Youtao Zhang, and Jun Yang. Simple virtual channel allocation for high throughput and high frequency on-chip routers. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–11, Jan 2010. doi: 10.1109/HPCA.2010.5416640.

[26] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, T. Mohsenin, D. Truong, and J. Cheung. Asap: A fine-grained many-core platform for dsp applications. *IEEE Micro*, 27(2):34–45, March 2007. ISSN 0272-1732. doi: 10.1109/MM.2007.29.

[27] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das. A case for dynamic frequency tuning in on-chip networks. In *Proc. MICRO-42*, pages 292–303, 2009.

[28] Liang Guang, Ethiopia Nigussie, Lauri Koskinen, and Hannu Tenhunen. Autonomous DVFS on supply islands for energy-constrained NoC communication. In *Proc. ARCS 2009*, volume 5455 of *Lect. Notes Comput. Sc.*, pages 183–194. 2009.

[29] Li Shang, Li-Shiuan Peh, and N.K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proc. HPCA*, pages 91–102, 2003. doi: 10.1109/HPCA.2003.1183527.

[30] Jia Zhan, Nikolay Stoimenov, Jin Ouyang, Lothar Thiele, Vijaykrishnan Narayanan, and Yuan Xie. Optimizing the NoC slack through voltage and frequency scaling in hard real-time embedded systems. 33(11):1632–1643, November 2014. ISSN 0278-0070. doi: 10.1109/TCAD.2014.2347921.

[31] Robert Hesse and Natalie Enright Jerger. Improving DVFS in NoCs with coherence prediction. In *Proc. NOCS*, pages 24:1–24:8, 2015. ISBN 978-1-4503-3396-2.

[32] X. Wang, Tengfei Wang, T. Mak, M. Yang, Y. Jiang, and M. Daneshtalab. Fine-grained runtime power budgeting for networks-on-chip. In *Proc. ASPDAC*, pages 160–165, 2015. doi: 10.1109/ASPDAC.2015.7058998.

[33] Wonyoung Kim, D.M. Brooks, and Gu-Yeon Wei. A fully-integrated 3-level dc/dc converter for nanosecond-scale dvs with fast shunt regulation. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 268–270, Feb 2011. doi: 10.1109/ISSCC.2011.5746313.

[34] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar. A 2 Tb/s 6x4 mesh network for a single-chip cloud computer with DVFS in 45 nm CMOS. 46(4):757–766, Apr. 2011.

[35] Xi Chen, Zheng Xu, Hyungjun Kim, Paul Gratz, Jiang Hu, Michael Kishinevsky, and Umit Ogras. In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches. *ACM Trans. on Design Automation of Electronic Systems*, 18(4):1–21, Oct. 2013. ISSN 10844309. doi: 10.1145/2504905.

[36] Xi Chen, Zheng Xu, Hyungjun Kim, Paul V. Gratz, Jiang Hu, Michael Kishinevsky, Umit Ogras, and Raid Ayoub. Dynamic voltage and frequency scaling for shared resources in multicore processor designs. In *Proc. DAC*, pages 114:1–114:7, 2013.

[37] Jae-Yeon Won, Xi Chen, P. Gratz, Jiang Hu, and V. Soteriou. Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management. In *Proc. HPCA*, pages 308–319, 2014. doi: 10.1109/HPCA.2014.6835941.

[38] Andrea Bianco, Paolo Giaccone, Mario Roberto Casu, and Nanfang Li. Exploiting space diversity and dynamic voltage frequency scaling in multiplane network-on-chips. In *Proc. GLOBECOM*, pages 3080–3085. IEEE, 2012.

[39] Jörg Henkel, Haseeb Bukhari, Siddharth Garg, Muhammad Usman Karim Khan, Heba Khdr, Florian Kriebel, Umit Ogras, Sri Parameswaran, and Muhammad Shafique. Dark silicon: From computation to communication. In *Proc. NOCS*, pages 23:1–23:8, 2015. ISBN 978-1-4503-3396-2. doi: 10.1145/2786572.2788707.

[40] Manoj Kumar Yadav, Mario Roberto Casu, and Maurizio Zamboni. LAURA-NoC: Local automatic rate adjustment in network-on-chips with a simple DVFS. 60(10): 647–651, Oct. 2013.

[41] A. Samih, R. Wang, A. Krishna, C. Maciocco, C. Tai, and Y. Solihin. Energy-efficient interconnect via router parking. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 508–519, Feb 2013. doi: 10.1109/HPCA.2013.6522345.

[42] H. Matsutani, M. Koibuchi, D. Wang, and H. Amano. Adding slow-silent virtual channels for low-power on-chip networks. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 23–32, April 2008. doi: 10.1109/NOCS.2008.4492722.

[43] L. Chen and T. M. Pinkston. Nord: Node-router decoupling for effective power-gating of on-chip routers. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 270–281, Dec 2012. doi: 10.1109/MICRO.2012.33.

[44] H. Farrokhbakht, M. Taram, B. Khaleghi, and S. Hessabi. Toot: an efficient and scalable power-gating method for noc routers. In *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, Aug 2016. doi: 10.1109/NOCS.2016.7579326.

[45] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, Sept 2007. ISSN 0272-1732. doi: 10.1109/MM.2007.4378783.

[46] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano, and Daihan Wang. Runtime power gating of on-chip routers using look-ahead routing. In *2008 Asia and South Pacific Design Automation Conference*, pages 55–60, March 2008. doi: 10.1109/ASPDAC.2008.4484015.

[47] L. Chen, D. Zhu, M. Pedram, and T. M. Pinkston. Power punch: Towards non-blocking power-gating of noc routers. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 378–389, Feb 2015. doi: 10.1109/HPCA.2015.7056048.

[48] Alberto Ghiribaldi, Daniele Ludovici, Michele Favalli, and Davide Bertozzi. System-level infrastructure for boot-time testing and configuration of networks-on-chip with programmable routing logic. In *VLSI-SoC*, pages 308–313. IEEE, 2011. ISBN 978-1-4577-0171-9. URL `http://dblp.uni-trier.de/db/conf/vlsi/vlsisoc2011.html#GhiribaldiLFB11`.

[49] T. Nachiondo, J. Flich, and J. Duato. Destination-based hol blocking elimination. In *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, volume 1, pages 10 pp.–, 2006. doi: 10.1109/ICPADS.2006.34.

[50] Weichen Liu, Jiang Xu, Xiaowen Wu, Yaoyao Ye, Xuan Wang, Wei Zhang, M. Nikdast, and Zhehui Wang. A noc traffic suite based on real applications.

In *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pages 66 –71, july 2011. doi: 10.1109/ISVLSI.2011.49.

[51] Nangate freepdk45 generic open cell library ver 1.0, February 2008. URL `http://www.si2.org/openeda.si2.org/projects/nangatelib/`.

[52] T. Chelcea and S.M. Nowick. A low-latency fifo for mixed-clock systems. In *VLSI, 2000. Proceedings. IEEE Computer Society Workshop on*, pages 119–126, 2000. doi: 10.1109/IWV.2000.844540.

[53] D. Marculescu and P. Choudhary. Hardware based frequency/voltage control of voltage frequency island systems. In *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, pages 34–39, Oct 2006. doi: 10.1145/1176254.1176265.

[54] Mario R. Casu and Paolo Giaccone. Rate-based vs delay-based control for dvfs in noc. In *Proc. DATE*, pages 1096–1101, 2015.

[55] D.E. Lackey, P.S. Zuchowski, T.R. Bednar, D.W. Stout, S.W. Gould, and J.M. Cohn. Managing power and performance for system-on-chip designs using voltage islands. In *Proc. ICCAD*, pages 195–202, 2002. doi: 10.1109/ICCAD.2002.1167534.

[56] U.Y. Ogras, R. Marculescu, D. Marculescu, and Eun Gu Jung. Design and management of voltage-frequency island partitioned networks-on-chip. 17(3):330–341, March 2009. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2011229.

[57] A.B. Kahng, B. Lin, and S. Nath. Orion3.0: A comprehensive noc router estimation tool. *Embedded Systems Letters, IEEE*, 7(2):41–45, June 2015. ISSN 1943-0663. doi: 10.1109/LES.2015.2402197.

[58] M.R. Casu, M.K. Yadav, and M. Zamboni. Power-gating technique for network-on-chip buffers. *Electronics Letters*, 49(23):1438–1440, Nov 2013. ISSN 0013-5194. doi: 10.1049/el.2013.3225.

[59] J.V. Escamilla, J. Flich, and P.J. Garcia. Head-of-line blocking avoidance in networks-on-chip. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 796–805, May 2013. doi: 10.1109/IPDPSW.2013.214.

[60] Intel Corp. Xeon phi. Available at `http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner?wapkw=knight+corner`, .

[61] Intel Corp. The single-chip cloud computer. Available at `http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html`, .

[62] M. Jurczyk and T. Schwederski. Phenomenon of higher order head-of-line blocking in multistage interconnection networks under nonuniform traffic patterns, 1996.

[63] Marcello Coppola, Miltos D. Grammatikakis, Riccardo Locatelli, Giuseppe Maruccia, and Lorenzo Pieralisi. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2008. ISBN 1420044710, 9781420044713.

[64] Ravi Iyer, Li Zhao, Fei Guo, Ramesh Illikkal, Srihari Makineni, Don Newell, Yan Solihin, Lisa Hsu, and Steve Reinhardt. QoS policies and architecture for cache/memory in CMP platforms. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):25, June 2007. ISSN 01635999. doi: 10.1145/1269899.1254886. URL `http://portal.acm.org/citation.cfm?doid=1269899.1254886`.

[65] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das. Application-aware prioritization mechanisms for on-chip networks. *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, page 280, 2009. doi: 10.1145/1669112.1669150. URL `http://portal.acm.org/citation.cfm?doid=1669112.1669150`.

[66] Boris Grot, S.W. Keckler, and O. Mutlu. Preemptive virtual clock: a flexible, efficient, and cost-effective QOS scheme for networks-on-chip. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 268–279. ACM, 2009. ISBN 9781605587981. URL `http://portal.acm.org/citation.cfm?id=1669149`.

[67] A. Banerjee and S.W. Moore. Flow-aware allocation for on-chip networks. In *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 183 –192, may 2009. doi: 10.1109/NOCS.2009.5071466.

[68] Dong Wu, Bashir M. Al-Hashimi, and Marcus T. Schmitz. Improving routing efficiency for network-on-chip through contention-aware input selection. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 36–41, Piscataway, NJ, USA, 2006. IEEE Press. ISBN 0-7803-9451-8. doi: http://dx.doi.org/10.1145/1118299.1118310. URL `http://dx.doi.org/10.1145/1118299.1118310`.

[69] T Marescaux, A. Rangevall, V Nollet, A Bartic, and H Corporaal. Distributed congestion control for packet switched networks on chip. In *Parallel Computing: Current Future Issues of High-End Computing, Proceedings of the International Conference ParCo*, volume 33, pages 761–768. Citeseer, 2005. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.1586&rep=rep1&type=pdf`.

[70] Ming Li, Qing-An Zeng, and Wen-Ben Jone. Dyxy: a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 849–852, New York, NY, USA, 2006. ACM. ISBN 1-59593-381-6. doi: http://doi.acm.org/10.1145/1146909.1147125. URL `http://doi.acm.org/10.1145/1146909.1147125`.

[71] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, and Li-Shiuan Peh. Research challenges for on-chip interconnection networks. *Micro, IEEE*, 27(5):96 –108, sept.-oct. 2007. ISSN 0272-1732. doi: 10.1109/MM.2007.4378787.

[72] R. Marculescu, U.Y. Ogras, Li-Shiuan Peh, N.E. Jerger, and Y. Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(1):3 –21, jan. 2009. ISSN 0278-0070. doi: 10.1109/TCAD.2008.2010691.

[73] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38, June 2006. ISSN 0360-0300. doi: http://doi.acm.org/http://doi.acm.org/10.1145/1132952.1132953. URL `http://doi.acm.org/http://doi.acm.org/10.1145/1132952.1132953`.

[74] M. Millberg and A. Jantsch. Priority based forced requeue to reduce worst-case latencies for bursty traffic. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1070 –1075, april 2009.

[75] H. Kim, Y. Kim, and J. Kim. Clumsy flow control for high-throughput bufferless on-chip networks. *Computer Architecture Letters*, PP(99):1, 2012. ISSN 1556-6056. doi: 10.1109/L-CA.2012.22.

[76] Ernst Gunnar Gran, Magne Eimot, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, Lars Paul Huse, and Gilad Shainer. First experiences with congestion control in infiniband hardware. In *IPDPS*, pages 1–12, 2010.

[77] J.V. Escamilla, J. Flich, and P.J. Garcia. Icaro: Congestion isolation in networks-on-chip. In *Proc. NoCS*, pages 159–166, 2014.

[78] Kalray. Kalray, mppa-256 bostan, 2014. URL `http://www.kalrayinc.com/kalray/products`.

[79] N.P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*, pages 364–373, May 1990. doi: 10.1109/ISCA.1990.134547.

[80] Rui Hou, Lixin Zhang, M.C. Huang, Kun Wang, H. Franke, Yi Ge, and Xiaotao Chang. Efficient data streaming with on-chip accelerators: Opportunities and challenges. In *Proc. HPCA*, pages 312–320, 2011. doi: 10.1109/HPCA.2011.5749739.

[81] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. Cost-efficient on-chip routing implementations for cmp and mpsoc systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(4):534–547, April 2011. ISSN 0278-0070. doi: 10.1109/TCAD.2011.2119150.

[82] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1855840.1855861`.

[83] Wonyoung Kim, M.S. Gupta, Gu-Yeon Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134, Feb 2008. doi: 10.1109/HPCA.2008.4658633.

[84] C.A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, M.S. Yousif, and C.R. Das. Vichar: A dynamic virtual channel regulator for network-on-chip routers. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 333–346, Dec 2006. doi: 10.1109/MICRO.2006.50.

[85] José V. Escamilla, José Flich, and Pedro Javier García. *Efficient DVFS Operation in NoCs Through a Proper Congestion Management Strategy*, pages 339–351. Springer International Publishing, Cham, 2015. ISBN 978-3-319-27308-2. doi: 10.1007/978-3-319-27308-2_28. URL `http://dx.doi.org/10.1007/978-3-319-27308-2_28`.

[86] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002. ISSN 0018-9162. doi: 10.1109/2.976921.

[87] J. V. Escamilla, M. R. Casu, and J. Flich. Increasing the efficiency of latency-driven dvfs with a smart noc congestion management strategy. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 241–248, Sept 2016. doi: 10.1109/MCSoC.2016.42.

[88] S.S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The alpha 21364 network architecture. In *Hot Interconnects 9*, pages 113–117, 2001. doi: 10.1109/HIS.2001.946702.

[89] A. Bakhoda, J. Kim, and T.M. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *43rd IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, pages 421–432, Dec 2010. doi: 10.1109/MICRO.2010.50.

[90] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Express virtual channels: Towards the ideal interconnection fabric. *SIGARCH Comput. Archit. News*, 35(2):150–161, June 2007. ISSN 0163-5964. doi: 10.1145/1273440.1250681.

[91] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and J. Plosila. Q-learning based congestion-aware routing algorithm for on-chip network. In *Proc. NESEA*, pages 1–7, 2011. doi: 10.1109/NESEA.2011.6144949.

[92] M. S. Sayed, A. Shalaby, M. El-Sayed Ragab, and V. Goulart. Congestion mitigation using flexible router architecture for network-on-chip. In *Electronics, Communications and Computers (JEC-ECC), 2012 Japan-Egypt Conference on*, pages 182–187, March 2012. doi: 10.1109/JEC-ECC.2012.6186980.