IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Worst-Case Communication Time Analysis for On-Chip Networks with Finite Buffers

RUBENS VICENTE DE LIZ BOMER [1], CESAR ALBENES ZEFERINO [1] LAIO ORIEL SEMAN [1], VALDERI REIS QUIETINHO LEITHARDT [2,3],

[1]Laboratory of Embedded and Distributed Systems, University of Vale do Itajaí, Itajaí 88302-901, Brazil
[2]COPELABS - Lusófona University of Humanities and Technologies, Campo Grande 376, 1749-024 Lisboa, Portugal
[3]VALORIZA, Research Center for Endogenous Resources Valorization, Instituto Politécnico de Portalegre, 7300-555 Portalegre, Portugal

Corresponding author: Cesar Albenes Zeferino (e-mail: zeferino@univali.br).

This work was supported by Comissão de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Finance Code 001, Conselho Nacional de Desenvolvimento Científico e Tecnologico (CNPq) – Processes 436.982/2018-8, 313.513/2021-0 and 140.368/2021-3, National Funds through the Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) by the Project "VALORIZA—Research Centre for Endogenous Resource Valorization" under Grant UIDB/05064/2020 and Grant UIDB/04111/2020.

ABSTRACT Network-on-Chip (NoC) is the ideal interconnection architecture for many-core systems due to its superior scalability and performance. An NoC must deliver critical messages from a real-time application within specific deadlines. A violation of this requirement may compromise the entire system operation. Therefore, a series of experiments considering worst-case scenarios must be conducted to verify if deadlines can be satisfied. However, simulation-based experiments are time-consuming, and one alternative is schedulability analysis. In this context, this work proposes a schedulability analysis to accelerate design space exploration in real-time applications on NoC-based systems. The proposed worst-case analysis estimates the maximum latency of traffic flows assuming direct and indirect blocking. Besides, we consider the size of buffers to reduce the analysis' pessimism. We also present an extension of the analysis, including self-blocking. We conduct a series of experiments to evaluate the proposed analysis using a cycle-accurate simulator. The experimental results show that the proposed solution presents tighter results and runs four orders of magnitude faster than the simulation.

INDEX TERMS Network-on-Chip, timing analysis, design space exploration.

I. INTRODUCTION

In the early 2000s, researchers began to look for a reusable and scalable interconnection architecture [1], [2], [3], [4], [5]. The solution is an integrated network on a single chip, named Network-on-Chip or NoC [2]. Such architecture can support multiple simultaneous communications, and its performance increases with the system size, which does not occur in the shared bus architecture.

The study of NoCs is still a current topic of interest. Researchers in [6] suggest examining networks' structure and behavior linking citations, subjects, and co-authors to understand how NoC research has developed over the past two decades. Some scientists are also looking into using machine learning with NoC. In [7], the authors propose a method that utilizes machine learning to improve the energy efficiency and performance of NoC systems. They propose using deep reinforcement learning to adjust the voltage and frequency of NoC routers and links and using neural networks to control distributed agents. This method was tested using real-world and simulated data and found to improve energy and delay by 30-40% compared to non-machine learning methods and by 8% compared to other machine learning methods. This method can adjust and handle large systems, making it useful for preventing issues such as overheating.

Among the current research topics in NoCs, is energy consumption analysis. In [8], energy-efficient scheduling algorithms for Multi-Processor System-on-Chip (MPSoC)-based NoC systems are proposed. These algorithms minimize execution time, energy consumption, and network load. They are

beneficial for scheduling large scientific tasks with hundreds or thousands of subtasks. The results of the experiments show that these proposed algorithms significantly reduce the execution time. These techniques are particularly relevant for high-performance computing systems, where real-world scientific applications are becoming more complex and demanding more time and resources. However, the utilization-centric design approach of NoCs can raise security concerns, such as unauthorized access or modification of intermediate routers, due to the sharing of hardware among all the IP (Intellectual Property) blocks. This sharing can compromise the integrity and confidentiality of the data being routed [9].

Just as significant as addressing security concerns, schedulability analysis is crucial in assessing the timing properties of real-time systems. It involves evaluating periodic or sporadic tasks, their worst-case execution time, and a scheduling policy to determine if it is possible to schedule them without missing any deadlines. Real-time (RT) applications have specific communication requirements which depend on the execution time of the service. All the communication packets must reach the destination within a pre-established time (i.e., the deadline). Even in worst-case scenarios, failure to meet a deadline for hard RT applications can have severe consequences [10].

In [11], the authors propose a method for enhancing the schedulability of real-time embedded systems using an MPSoC with an NoC communication paradigm, taking into account the specialized architecture and routing strategy needed by NoC to meet the strict deadlines of real-time systems. To further improve end-to-end schedulability, the authors consider all possible minimal paths and include deadlock avoidance in the routing strategy rather than relying on fixed XY routing. In addition, NoCs must be able to provide different levels of service for various applications in the same network, known as Quality-of-Service (QoS) [5].

The services offered by the NoC must show predictable behavior to provide RT communications in Systems-on-Chip (SoCs). However, the NoC's behavior is partially non-deterministic, as contentions can occur due to competition for shared resources, such as buffers and communication channels. This issue leads to delays in packet delivery and deadline losses. To mitigate this problem, priority-based arbitration mechanisms, flow control with support for virtual channels, and an analytical approach to predict the fulfillment of all timing constraints should be used [12]. The design space of NoCs is ample and involves a set of different parameters whose combination impacts the costs and performance of the network and the system. Synthesis and simulation tools aid in obtaining these metrics, but they have a high computational cost. Therefore, the exploration of the design space of NoCs takes time. Nonetheless, a system designer can accelerate this exploration using an analytical approach that allows obtaining an initial estimate of the performance of different configurations, reducing the number of configurations to be analyzed with synthesis and simulation tools [13].

Considering the aspects mentioned above, schedulability analysis is particularly important for NoCs as it helps to determine whether the traffic flows can meet their deadlines in congestion and contention for shared resources. In addition, designers can use it to evaluate the performance of different NoC configurations, such as the number of virtual channels, the size of buffers, and the routing algorithm, and to identify potential problems with meeting deadlines. Various techniques can be used for schedulability analysis, including exact analysis, approximate analysis, and simulation. The detailed analysis involves analyzing the system to determine each traffic flow's worst-case response time. In contrast, the approximate analysis uses simplified models to obtain a rough estimate of the response time. Finally, simulation involves running a model of the system to observe the behavior of traffic flows under different conditions.

In this context, we present the schedulability analysis of a wormhole switching NoC with XY routing and round-robin arbiters, considering traffic flow interferences. The main contribution of this work is the reduction of the analysis pessimism by considering the size of buffers. In addition, we present an extension to consider the self-blocking effect, a characteristic explored by a few works in the literature.

The remainder of this paper is organized into seven sections. Section 2 discusses state of the art on schedulability analysis for NoC-based systems. Section 3 describes the architecture of the reference NoC, while Section 4 presents the system model and assumptions. The proposed schedulability analysis is described in Section 5. Section 6 summarizes the experimental results, and Section 7 presents the final remarks.

II. RELATED WORK

Schedulability analysis is a technique used to evaluate the performance of computer systems in real-time environments. Researchers have proposed various methods for conducting schedulability analysis in NoCs using different types of arbitration.

For example, in [14], the authors proposed a method for analyzing computer systems using schedulability analysis. Later, in [12], the authors suggested using this technique for real-time systems with NoCs that use fixed-priority arbitration. They also introduced a way to predict the network latency caused by higher-priority traffic flows. In [15], the authors expanded on this method to include self-blocking contention. In [16], the researchers continued investigating fixed-priority-based networks and presented schedulability tests that account for each task's end-to-end latency and dependencies. In [17], the authors presented methods to reduce the pessimism of schedulability analysis in fixed-priority-based networks by only considering traffic flow interference in specific network sections where direct and indirect blockings occur.

Other studies have applied schedulability analysis to NoCs that use dynamic arbiters. For instance, in [18], the authors used static analysis to estimate the worst-case communication latency in an NoC that uses a Hop-based arbiter. This

approach uses a weighted graph to identify the traffic flow that causes the worst latency. In [19], the authors proposed using an Earliest-Deadline-First (EDF) policy in NoCs with wormhole switching. They also presented a contention analysis to estimate the worst-case latency of individual flows, considering both direct and indirect interference. In [20], the authors presented a schedulability analysis for real-time communications in wormhole networks using a round-robin arbitration scheme with multiple stages. This work was later extended in [21] to include a best-effort NoC.

In [22], the authors proposed analytical models to estimate the worst-case performance of a 2D mesh NoC with round-robin arbiters and two routing algorithms: a deterministic (XY) and an adaptive (negative-first). The model considers the free buffer space in its contention analysis, but the NoC does not use virtual channels and sets the buffer size to three flits to generate worst-case situations. In [23], the authors proposed three properties to identify worst-case scenarios and reduce the pessimism of schedulability analysis. The first property determines the worst-case scenario when computing the maximum blocking delay that a flow can suffer using round-robin arbiters. The second property determines the maximum blocking delay a flow can suffer due to unblocking. The third property determines when an indirect flow influences the transmission delay of the analyzed flow by taking into account the size of the buffer. However, the study assumes only one-flit buffers and no virtual channels.

Based on the literature review summarized in Table 1, this work proposes the schedulability analysis of a wormhole switching NoC with XY routing, round-robin arbitration, and virtual channels with traffic flow preemption, considering the interference of direct and indirect blocking. In addition, the analysis considers the size of buffers to reduce the pessimism and extends the models, including self-blocking. This work offers a more comprehensive approach to schedulability analysis for NoCs, addressing a wider range of factors and scenarios than previous works.

III. NETWORK ARCHITECTURE

This work utilizes as reference a wormhole switching NoC named SoCIN-Q [24], which uses a 2-D mesh topology (Fig. 1) and wormhole switching. Each router of SoCIN-Q has up to five communication ports named: L (Local), N (North), E (East), S (South), and W (West). The Local port connects a core to the network, and the others interconnect the router with its neighbors. Each of these ports has a pair of input and output modules (or channels) connected to a crossbar and implements the logic necessary for packet forwarding.

Wormhole switching is a technique used in NoCs (Network-on-Chips) to improve the network's performance. It allows the transmission of data packets or flits through the network in a continuous stream rather than sending each flit individually. This approach reduces the latency and improves the overall performance of the network.

In SoCIN, each input channel of a router has a buffer that

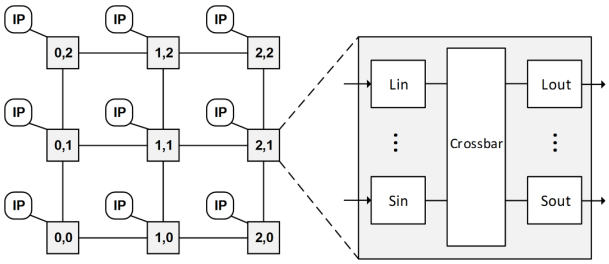


FIGURE 1: A SoCIN-Q with a 3×3 2D-mesh topology and the router internal structure

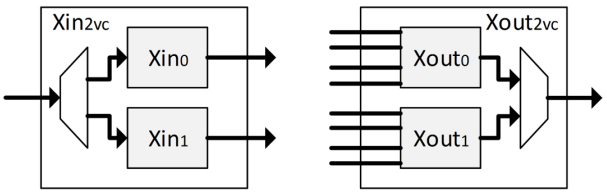


FIGURE 2: The input and output modules of SoCIN-Q router, each module with two virtual channels (VC_0 and VC_1)

stores incoming data. The network uses credit-based flow control to regulate flit-level data transfer and deal with the limited capacity of these buffers (a flit is the piece of data over which the flow control is done). Applying this technique, the output channel of the sending router uses a credit counter initialized with the capacity of the receiving router's buffer. This counter is decremented when a packet flit is sent and incremented when the receiving router sends back credit. A credit signals that a flit was forwarded and a slot was freed in the receiving router's input buffer. It is worth noting that the sending router can only send a flit if the credit is greater than 0. Limiting the amount of data that can be sent at once helps keep the network from getting congested.

Fig. 2 depicts the internal organization of the input and output modules, named Xin2VC and Xout2VC, where X is the port identifier. Each module has two virtual channels (VCs) with different priority levels. VC_0 has the highest priority and can preempt VC_1 when necessary. The Xin modules are responsible for incoming flow control, memorization, and routing. The Xout modules schedule the use of the output channels and regulate the flow of the outgoing flits. These modules use round-robin arbiters to perform the output schedule.

For this study, we use a 2-D mesh wormhole NoC with credit-based flow control, XY routing, round-robin arbitration, and two virtual channel buffers at the input modules.

IV. SYSTEM MODEL

In this work, we assume a real-time system Γ mapped to an $m \times m$ SoCIN-Q composed of n traffic flows $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. A direct graph represents the NoC with ver-

TABLE 1: Literature review summary.

Reference	Schedulability Analysis for
[14]	computer systems
[12]	real-time system using NoC with fixed-priority arbitration
[15]	real-time system using NoC with fixed-priority arbitration (extended to include self-blocking contention)
[16]	real-time system using NoC with fixed-priority arbitration (considering end-to-end latency and task dependencies)
[17]	real-time system using NoC with fixed-priority arbitration (reducing pessimism by analyzing traffic flow interference in sections of respective paths)
[18]	real-time system using NoC with Hop-based arbiter
[19]	real-time system using NoC with Earliest-Deadline-First policy and wormhole switching
[20]	real-time system using NoC with round-robin arbitration and multiple stages
[21]	best-effort NoC with round-robin arbitration and multiple stages
[22]	2D mesh NoC with round-robin arbiter and two routing algorithms
[23]	NoC with round-robin arbiter (considering worst-case scenarios, unblocking delay, and influence of indirect flows)

tices and edges. Each vertex denotes a router R, while the edge E denotes a physical link. An edge $E_{a,b} = \{R_a \rightarrow R_b\}$ is the link from router R_a to router R_b. Each node connected to the network generates only one traffic flow. A set of links β_i denotes the path of each traffic flow τ_i, and β_i^{size} defines the number of links in the path. For instance, if a traffic flow crosses routers R_a, R_b, R_c, and R_d, then $\beta_i = \{E_{a,b}, E_{b,c}, E_{c,d}\}$ and β_i^{size} equals 3.

Each traffic flow τ_i is characterized by a set of attributes $\{L_i^{\max}, p_i, d_i, e_i, v_i\}$, where: $L_i^{\max}$ is the maximum latency (*i.e.* the worst-case response time), p_i defines the time interval between successive packets (the period), d_i represents the packet deadline, e_i denotes the packet service time, and v_i is the identifier of the virtual channel. We assume that the traffic flow period p_i includes the task processing time and the delay to inject a packet into the NoC. Initially, we consider periodic traffic flows with a deadline less or equal to its period ($d_i \leq p_i \, \forall \, \tau_i \in \Gamma$). Later we present an extension of the analysis allowing $d_i > p_i$ and including self-blocking interferences. The service time e_i denotes the number of cycles necessary for a packet from τ_i to pass entirely through a router. Finally, v_0 is the highest priority channel.

In SoCIN-Q, the packet header experiences a constant service time H at each router. H is the number of clock cycles needed to store, route, and schedule a header flit. Assuming the NoC configuration presented in Section 3, two cycles are necessary to store the header at the input buffer and run the routing algorithm to request an output channel. The selected output module spends the third cycle scheduling the request (*i.e.* the arbitration). The payload flits follow the header in a pipelined way, spending one cycle per flit. Let f_i denote the packet payload size in flits, then the total service time of a τ_i packet is $e_i = H + f_i$.

The minimum latency (or basic latency) occurs when a traffic flow does not suffer any contention in the NoC. The routing path, the packet size, and the header service time define its value. The minimum latency L_i^{Min} of a traffic flow τ_i is:

$$L_i^{\min} = H \times (\beta_i^{\text{size}} + 1) + f_i + 1 \qquad (1)$$

where $(\beta_i^{\text{size}} + 1)$ expresses the number of routers in the path (*i.e.* the hop counting). An extra cycle considers the time the destination node spends to receive the header flit. Fig. 3.a presents an example of a traffic flow τ_1 that uses a path composed of two links ($\beta_1^{\text{size}} = 2$) and three routers. The minimum latency for a packet with a 4-flit payload ($f_1 = 4$) equals 14 cycles, as Fig. 3.b depicts.

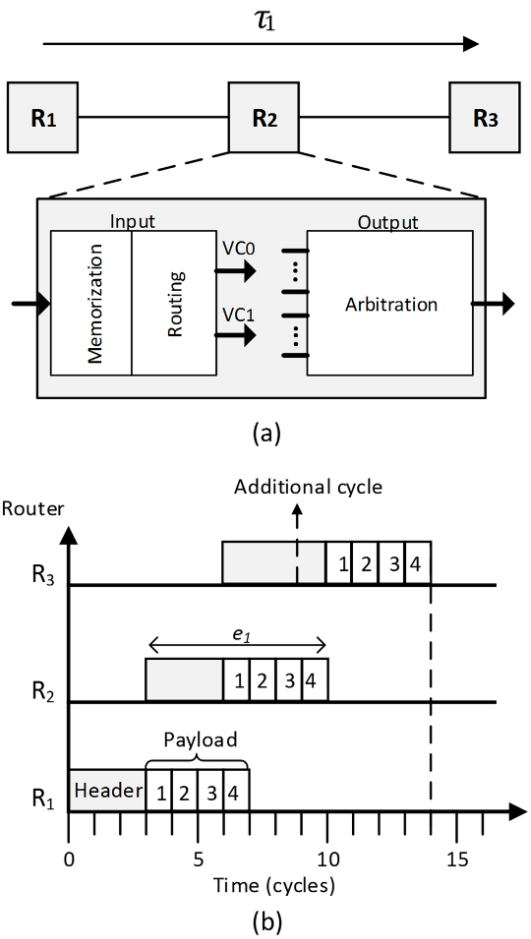


FIGURE 3: Example of traffic flow: (a) path; and (b) time diagram

A summary of the presented variables is presented in Table 2.

TABLE 2: Definition of variables.

Variable	Definition
Γ	Real-time system mapped to a 2D mesh NoC
τ_i	Traffic flow in the NoC
m	Dimension of the 2D mesh NoC
n	Number of traffic flows in the NoC
R	Router in the NoC
E	Physical link in the NoC
β_i	Path of traffic flow τ_i
β_i^{size}	Number of links in the path of traffic flow τ_i
$L_i^{\max}$	Maximum latency of traffic flow τ_i
p_i	Period of traffic flow τ_i
d_i	Packet deadline of traffic flow τ_i
e_i	Packet service time of traffic flow τ_i
v_i	Virtual channel identifier for traffic flow τ_i
H	Header service time at each router
f_i	Number of payload flits in a packet from traffic flow τ_i
$L_i^{\min}$	Minimum latency of traffic flow τ_i

V. NETWORK WORST-CASE ANALYSIS

A traffic flow packet competes for router ports while transmitting through the NoC. A packet using the virtual channel with the highest priority establishes communication with the requested output port. If another packet with the same priority level requests the port simultaneously, a round-robin arbiter chooses one of the requests. The round-robin principle is that a just-served request has the lowest priority on the next round of arbitration [24]. Because of the round-robin non-deterministic behavior, we define properties to simplify the Worst-Case Analysis (WCA) and reduce the number of scenarios to be explored. Based on similar works that used round-robin arbiters [20], [21], [23], we determine the worst-case scenario of a flow τ_i in a router R_k using the following definitions:

Definition 1. *The τ_i request for an output port always receives the lowest priority in the round-robin arbitration.*

Definition 2. *Every traffic flow τ_j that generates a blocking in τ_i arrives at the same time in R_k, while the other flows arrive immediately after the release of the output port, before the next round of arbitration.*

Fig. 4 shows three traffic flows sharing the same virtual channel. The worst-case scenario of τ_1 occurs when its packet header arrives at R_2. Assuming Definition 1, τ_1 receives the lowest priority to request the output port. As stated by Definition 2, τ_2 arrives at the same time in R_2 and requests the same output port that τ_1 is requesting, consequently blocking τ_1. When the output port is released, τ_1 is blocked by τ_3, which arrives immediately after the τ_2 transmission ends.

By considering the presented definitions, we will determine the worst-case latency considering direct blocking, indirect blocking, and self-blocking. The following subsections cover each interference separately.

A. DIRECT BLOCKING

Direct blocking can occur when traffic flows have common links in their paths. If $\beta_i \cap \beta_j \neq \varnothing$ and the condition $v_i \geq v_j$ is true, then τ_j generates a direct blocking and τ_i must wait for its transmission over the shared link. Every traffic flow τ_j that fulfills these conditions belongs to the set of direct interferences S_i^D of τ_i. In the example of Fig. 4, link $E_{2,3}$ is in the path of all traffic flows, and, therefore, direct blocking can occur. If $v_1 \geq v_2$, then $\tau_2 \in S_1^D$ and directly blocks τ_1.

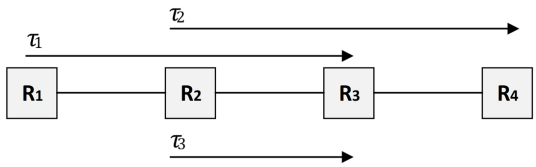


FIGURE 4: Example illustrating the worst-case scenario using Definitions 1 and 2

Theorem 1. *Assuming only direct blocking by traffic flows using the same virtual channel ($\tau_j \in S_i^D, v_i = v_j$), the maximum latency of a traffic flow τ_i is:*

$$L_i^{\min} + \sum_{\forall\, \tau_j \in S_i^D \wedge v_i = v_j} e_j \tag{2}$$

Proof. According to Definition 2, each traffic flow $\tau_j \in S_i^D$ arrives simultaneously with τ_i at a router and requests the same output port. Also, τ_i always receives the lowest priority in the round-robin arbitration, as stated by Definition 1. Then, τ_i is blocked and needs to wait for the service time e_j of each $\tau_j \in S_i^D$. However, traffic flows sharing the same virtual channel ($v_i = v_j$) can only block τ_i once since, in this case, there is no flow preemption. ■

If τ_j uses a virtual channel with a higher priority ($v_i > v_j$), τ_i can be preempted. According to the response-time test presented in [25], on uniprocessor systems, preemption can occur multiple times depending on the periodicity and the maximum execution time of the tasks. Equation (3) presents an adapted version of the response-time test for NoCs proposed in [12], where the number of preemptions interferences I_j generated by τ_j is determined by its period and by the maximum latency $L_i^{\max}$ of τ_i.

$$I_j = \left\lceil \frac{L_i^{\max}}{p_j} \right\rceil \tag{3}$$

In [17], one of the approaches to reduce the pessimism in (3) was to take into account the latency experienced in the common edges of τ_i and τ_j instead of the maximum latency. In this work, we proposed a simplified version of this approach using the following theorem:

Theorem 2. *The number of interferences I_j generated by a traffic flow $\tau_j \in S_i^D$ using a virtual channel with higher priority ($v_i > v_j$) is:*

$$I_j = \left\lceil \frac{sl(\tau_i, \tau_j) \times e_i + e_j}{p_j} \right\rceil \tag{4}$$

Proof. A traffic flow $\tau_j \in S_i^D$ using a higher priority virtual channel ($v_i > v_j$) can preempt τ_i to request an output port. However, τ_i can only be preempted while its packet flits use links in the path of τ_j. The number of interference I_j depends on the transmission time of τ_i through the shared links and the period of τ_j. Let $sl(\tau_i, \tau_j)$ be the number of shared links between τ_i and τ_j. A packet spends its service time e_i in each of those links. Also, in the worst-case scenario, τ_j is already blocking τ_i, and e_j must be considered. ■

Based on Theorem 1 and Theorem 2, the latency L_i^{Direct} generated only by direct blocking is computed by:

$$L_i^{\text{Direct}} = \sum_{\forall\, \tau_j \in S_i^D \wedge v_i = v_j} e_j + \sum_{\forall\, \tau_j \in S_i^D \wedge v_i > v_j} I_j \times e_j - 2 \quad (5)$$

where two cycles are discounted from the delay generated by traffic flows using higher-priority virtual channels. Since τ_i does not use the same virtual channel, it is only necessary to wait for the memorization cycles.

In summary, direct blocking occurs when two traffic flows share a link, and the flow with the lower priority virtual channel must wait for the flow with the higher priority to be transmitted. The maximum latency of a traffic flow is determined by the sum of the service time of all other flows that directly block it. Preemption, where a flow with a higher priority virtual channel can interrupt a flow with a lower priority, can also occur and is determined by the period and maximum latency of the interrupted flow. The number of times preemption occurs is determined by the common latency experienced by the two flows and the service time of the preempting flow. Finally, the system can also be subject to indirect blocking, where a flow blocks another flow indirectly through a chain of interfering flows. In this case, the maximum latency of a flow is determined by the sum of the service time of all blocking flows.

B. INDIRECT BLOCKING

A traffic flow τ_i can suffer an indirect blocking when a flow τ_k that has no links in common with τ_i (*i.e.* $\beta_i \cap \beta_k = \varnothing$) blocks directly τ_j ($\tau_j \in S_i^D$) and $v_i = v_j \geq v_k$. In the worst case, both τ_i and τ_j must wait for the transmission of τ_k. However, indirect blocking is not limited to situations with only three traffic flows. For instance, Fig. 5 depicts five traffic flows using the same virtual channel in a congestion state. The traffic flow τ_1 is directly blocked by τ_2 and indirectly blocked by τ_3, since $\beta_1 \cap \beta_2 \neq \varnothing$, $\beta_2 \cap \beta_3 \neq \varnothing$ and $\beta_1 \cap \beta_3 = \varnothing$. Every direct and indirect blocking affecting τ_3 also indirectly affects τ_1. Then, in the worst-case scenario, τ_1 suffers the delay generated by the indirect blocking of τ_4 and τ_5. Such situations are easily detected by verifying each traffic flow blocking τ_1.

In this subsection, we identify properties to reduce the pessimism of indirect blocking and consider the size of the buffers in the WCA.

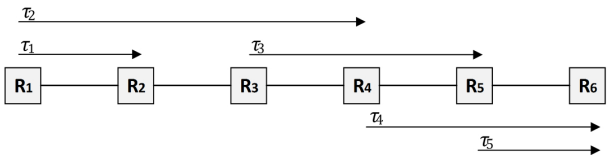


FIGURE 5: Example illustrating indirect blocking

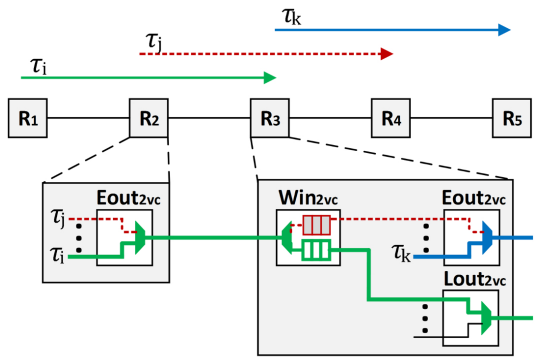


FIGURE 6: An example where $v_i > v_j \geq v_k$ and $\tau_k \notin S_i^I$

Property 1. *A traffic flow τ_k ($\tau_k \in S_j^D, \beta_i \cap \beta_k = \varnothing$) generates an indirect blocking in τ_i only if τ_i and τ_j ($\tau_j \in S_i^D$) are using the same virtual channel ($v_i = v_j$).*

Proof. Assuming the scenario presented in Fig. 6, where $\tau_j \in S_i^D$ and $\tau_k \in S_j^D$, a direct blocking from τ_k in τ_j cannot affect τ_i if $v_i > v_j \geq v_k$. In this situation, the credit-based flow control allows a traffic flow using virtual channel 1 to send flits when the traffic flow using virtual channel 0 has no credits (i.e. the buffer is full). Since τ_k blocks τ_j, the buffer of virtual channel 0 in R_3 becomes full, and τ_j cannot send more flits. Then τ_i, which uses the virtual channel 1 and a different buffer from τ_j, is allowed to send flits to R_3, as Fig. 6 depicts. An indirect blocking would occur only when $v_i = v_j$, since both traffic flows use the same buffer in R_3. If the condition $v_i = v_j \geq v_k$ is true, τ_k indirectly interferes in τ_i. Every flow τ_k that fulfills these conditions belongs to the set of indirect interferences S_i^I of τ_i. ■

Property 2. *If there are not enough buffers to store the entire packet of τ_j in the path after the last router where τ_i was blocked, and the router where τ_j is blocked by τ_k ($\beta_i \cap \beta_k = \varnothing$), then, τ_k influences τ_i.*

Proof. The buffers depth, named $FIFO_{depth}$, and the packet size of τ_j ($f_j + 1$) directly affect the influence of an indirect blocking generated by τ_k on τ_i. Let $R_{i,j}^{Last}$ be the last router where τ_j blocks τ_i and $R_{j,k}^{Block}$ notes the first router where τ_j is blocked by τ_k. The flits of τ_j remain in the buffers along its path while blocked. If at least one flit is in $R_{i,j}^{Last}$, because there is not enough buffer space in the other routers, then τ_i suffers an indirect blocking from τ_k. The influence value N_k of τ_k is determined by (6), where $hops(R_x, R_y)$ is a function that returns the number of hops between two routers. If $N_k \leq$

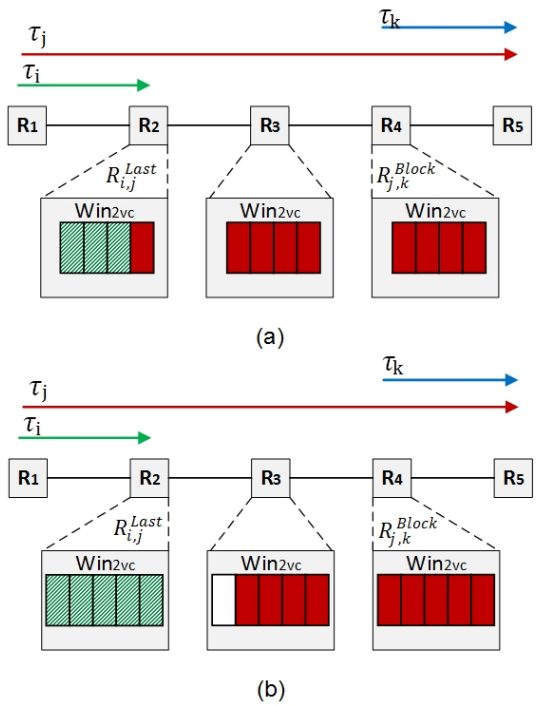


FIGURE 7: NoC configurations with buffers with (a) 4 slots and (b) 5 slots

0, *there is enough buffer in the path to store the packet of* τ_j *and* τ_k *does not influence* τ_i. *However, if* $N_k > 0$, *its value represents how many flits of* τ_j *are blocking* τ_i. *Since there are not enough buffers in the path to store the entire packet of* τ_j, τ_k *blocks indirectly* τ_i ($\tau_k \in S_i^I$). *If* $R_{i,j}^{Last} = R_{j,k}^{Block}$, *then* τ_k *is always influent, independent of the buffer size.*

$$N_k = f_j + 1 - \text{hops}(R_{i,j}^{\text{last}}, R_{j,k}^{\text{block}}) \times \text{FIFO}_{\text{depth}} \quad (6)$$

Fig. 7 depicts the worst-scenario of τ_i *in two NoC configurations, where* $v_i = v_j \geq v_k$ *and the packets of each traffic flow is 9-flit long, including the header. In the configuration of Fig. 7.a, each buffer has four slots for flits* ($FIFO_{depth} = 4$). *When* τ_j *is blocked by* τ_k *at* R_4, *its flits remain in the West buffers of* R_4, R_3, *and* R_2. *According to (6),* $N_k = 1$ *and its value represents the last flit of* τ_j *stored in* R_2, *which is blocking* τ_i *for the duration of the* τ_k *transmission. In Fig. 7.b, the buffer depth is incremented to 5 slots* ($FIFO_{depth} = 5$). *In this configuration, the entire packet of* τ_j *is stored in the West buffers of* R_3 *and* R_4. *Since the flits of* τ_j *are not blocking the West buffer of* R_2, τ_k *has no influence* ($N_k = -1$) *over* τ_i. ■

Assuming Property 1 and Property 2 to reduce the pessimism of the indirect blocking analysis, selecting only the influent flows to include in S_i^I, the latency of indirect blocking is computed by (7). Similar to the latency of direct blocking in (5), the number of interferences of τ_k is based on Theorem 2, since τ_i suffers the same delay generated by τ_k in τ_j. Based on (1), (5), and (7), the maximum latency $L_i^{\max}$ of a traffic flow τ_i is computed by (8).

$$L_i^{\text{indirect}} = \sum_{\forall \tau_k \in S_i^I \wedge v_i = v_k} e_k + \sum_{\forall \tau_k \in S_i^I \wedge v_i > v_k} I_k \times e_k - 2 \quad (7)$$

$$L_i^{\max} = L_i^{\min} + L_i^{\text{direct}} + L_i^{\text{indirect}} \quad (8)$$

Indirect blocking occurs when a traffic flow that does not share any links with another traffic flow still causes delays for it. This scenario can happen when the blocking traffic flow directly blocks a flow that the impacted flow depends on, and they share the same virtual channel. Properties are presented to reduce the indirect blocking analysis's pessimism and consider the buffers' size in the network-on-chip (NoC). The properties state that an indirect blocking will only occur when the blocking and impacted flows share the same virtual channel and that the size of the buffers in the NoC can limit the number of indirect blockings. A worst-case analysis method is also presented, which involves recursively identifying all the traffic flows causing indirect blockings for a particular flow. This method can be used to determine the maximum latency of a flow in an NoC with indirect blocking.

C. SELF-BLOCKING ANALYSIS

A traffic flow τ_i with $d_i \leq p_i$ does not have more than one packet transmitted simultaneously in the NoC unless a deadline violation occurs. If $d_i > p_i$, then multiple traffic flow packets can be transmitted through the NoC without missing their deadlines. In this case, self-blocking occurs when one or more packets block another packet of the same traffic flow. In this subsection, we present properties to determine the maximum latency of a traffic flow when self-blocking occurs.

In[15], the authors used the concept of busy period introduced by [14] to carry out the self-blocking analysis. In the context of NoCs, they defined the busy period as a contiguous time interval during which a set of links is kept busy by traffic flows using virtual channels with equal or higher priority than τ_i. Fig. 8 presents a time diagram of τ_i with four packets release (q_0, q_1, q_2, and q_3). If $L_i^{\max} > p_i$, packets are released while a previous instance q_n is already in the NoC. For instance, q_1 and q_2 are released before q_0 is delivered to their destination, represented by the first dotted down arrow. Equation (9) determines the number of queued packets in the worst-case scenario, noted by Q_i.

$$Q_i = \left\lceil \frac{L_i^{\max}}{p_i} \right\rceil \quad (9)$$

In this work, to simplify the self-blocking analysis and determine the maximum latency of a traffic flow τ_i, we assume the following additional definition:

Definition 3. *A queued packet of* τ_i *suffers the delay of every indirect blocking related to a direct blocking from a packet*

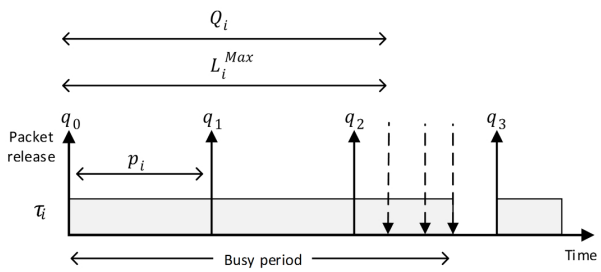


FIGURE 8: Busy period of traffic flow τ_i with multiple packets releases

of another traffic flow $\tau_j \in S_i^D$ that is released during its transmission.

A direct blocking τ_j and the related indirect blocking that occurs during the transmission of a queued packet from τ_i are considered a single blocking b_j and belong to the set of interferences S_i^{sb}. This single blocking has the attributes of τ_j with the extra delay of every indirect blocking τ_k that started from it. Let $d_i > p_i$, and assume $L_1^{\max} > p_1$ and $S_1^I = \{\tau_3, \tau_4, \tau_5\}$ in the example shown in Fig. 5. According to Definition 3, a queued packet from τ_1 is blocked by τ_2 and indirectly blocked by τ_3, τ_4 and τ_5 if a packet from τ_2 is released during its transmission. Then, τ_2 is considered as a single blocking $b_2 \in S_1^{sb}$ with the extra delay of τ_3, τ_4 and τ_5.

Theorem 3. *Assuming a traffic flow τ_i with $L_i^{\max} > p_i$, the maximum latency of the last queued packet instance, including self-blocking, is given by:*

$$L_i^{sb} = L_i^{\min} + Q_i \times e_i + \sum_{\forall\, b_j \in S_i^{sb}} e_{b_j} \times \left\lceil \frac{L_i^{sb}}{p_j} \right\rceil \qquad (10)$$

Proof. Let $L_i^{\max} > p_i$, so more than one packet is released during the worst-case scenario. We need to check the last queued packet to find the maximum latency, including self-blocking L_i^{sb}. We assume that the last packet of the queue is $q = Q_i$, and each queued packet released before q generates a self-blocking delay ($Q_i \times e_i$). If a blocking $b_j \in S_i^{sb}$ occurs during any of the queued packets transmission, it also affects the last queued packet. If $L_i^{sb} > L_i^{\max}$ and the number of queued packets is increased with L_i^{sb}, then a new iteration of (10), replacing $L_i^{\max}$ with L_i^{sb}, is necessary. ■

Property 3. *Assuming that the traffic flows are released only after the delivery of instance q_0 from τ_i, then a blocking b_j can only occur if its period is in L_i^{sb} and the number of interferences is no more than $Q_i - 1$ if $v_i = v_j$.*

Proof. We assume that the release of packets from blocking traffic flows starts after instance q_0 from τ_i reaches its destination. If $\frac{L_i^{sb}}{p_j} \geq 1$, then a blocking b_j can interfere in τ_i one or more times. Multiple packet releases from b_j can occur during L_i^{sb} and interfere in queued packets from τ_i. However, if $v_i = v_j$, the maximum number of interferences depends on the quantity of queued packets ($Q_i - 1$) since each packet released of b_j can block packets from τ_i only once. ■

Property 4. *If $L_i^{sb} > L_i^{\max}$ and there are not enough buffers in β_i to store all queued packets, then τ_i reaches its saturation point.*

Proof. The value of L_i^{sb} can increase at every iteration of (10) and consequently the number of queued packets. The NoC needs to have enough buffers to store every packet released by a traffic flow. Equation (11) determines the number of flits necessary to store all queued packets from τ_i, while (12) computes the total flits available in β_i. New packets are prevented from being injected in the NoC if $M_i^{Needed} > M_i^{Total}$ because there is a violation of the available buffer capacity in β_i. Also, the busy period of τ_i never ends if $L_i^{sb} > L_i^{\max}$, resulting in traffic flow saturation and a deadline violation. ■

$$M_i^{\text{needed}} = Q_i \times (f_i + 1) \qquad (11)$$

$$M_i^{\text{total}} = (\beta_i^{\text{size}} + 1) \times \text{FIFO}_{\text{depth}} \qquad (12)$$

In summary, direct blocking, indirect blocking, and self-blocking can affect the maximum latency of a traffic flow τ_i in a Network-on-Chip (NoC). Direct blocking occurs when two traffic flows with common links in their paths request the same output port simultaneously, and the one with the lower priority must wait for the other to finish transmitting. Indirect blocking occurs when a traffic flow blocks another traffic flow, which blocks a third traffic flow. Self-blocking occurs when a traffic flow with a period longer than its maximum latency has multiple packets being transmitted simultaneously in the NoC, and one or more packets block another packet of the same traffic flow. The maximum latency of a traffic flow can be calculated by taking into account the delays caused by direct and indirect blocking and self-blocking.

D. DISCUSSION

To simplify the Worst-Case Analysis (WCA) and reduce the number of scenarios to be explored, we define two properties: the first states that a traffic flow's request for an output port always receives the lowest priority in the round-robin arbitration, and the second states that every traffic flow that generates a blocking in another traffic flow arrives at the same time in a router, while the other flows arrive immediately after the release of the output port, before the next round of arbitration.

Then, three types of interference are considered: direct, indirect, and self-blocking. Direct blocking occurs when traffic flows have common links in their paths, and the traffic flow with the lower priority must wait for the transmission of the higher priority traffic flow over the shared link. Indirect blocking occurs when a traffic flow is blocked by another traffic flow that is not in its direct interference set but shares a

common link with a traffic flow in the direct interference set. Self-blocking occurs when a traffic flow is blocked by itself due to its deadline being greater than its period.

Finally, the section provides proof for each of the three types of interference, showing how to calculate the maximum latency for each case. The article also presents an extension of the analysis to allow for traffic flows with deadlines greater than their periods and includes self-blocking interference.

VI. EXPERIMENTAL RESULTS

This section evaluates the schedulability analysis for direct, indirect, and self-blocking. We implemented the proposed models in C++ and compared their results with the latencies obtained using a cycle-accurate SystemC-based NoC simulator named RedScarf [26]. The number of traffic flows and the size of the buffers varied according to the experiment. The NoC operates at 100 MHz in all simulations with the communication mechanisms defined according to the system model described in Section 4.

A. PESSIMISM REDUCTION

We conducted ten experiments in a 4×4 NoC with 16 periodic traffic flows, each sending 1000 packets. In each experiment, we randomly generate the traffic flows configuration. The payload size is selected from [4, 1000] flits, the deadline is selected from [40, 10000] cycles, and it equals its period. The traffic flows are randomly mapped onto the NoC, with one flow per node restriction. We compared the maximum latency of each traffic flow obtained through simulation with the latency estimated by the proposed WCA, considering the size of the buffers (named bWCA). In order to evaluate the pessimism reduction, we also applied the WCA without considering the buffers' size (nbWCA). We set the NoC buffers to their maximum capacity (1024 flits).

As Fig. 9.a shows, the bWCA model presents a lower average error than the nbWCA model in 6 out of 10 experiments. In these cases, the pessimism reduction is between 11% and 53%. The same occurs with the maximum error in experiments 3, 5, 6, 9, and 10, in which bWCA presents from 25% to 67% error reduction, as Fig. 9.b shows. The reason for such improvement is the Property 2 applied by bWCA, allowing the correct detection of indirect blocking influence. At the same time, the nbWCA model considers every indirect blocking as influent, increasing the computed maximum latency.

The experiments in which the average error is the same for both WCAs present indirect blocking in fewer quantities. In these configurations, some of the indirect blocking occurs when $R_{i,j}^{\text{last}} = R_{j,k}^{\text{block}}$ and the size of the buffers have no impact in its influence. In general, bWCA presents a tighter maximum latency with an average error of 10%, 5% less than nbWCA.

The maximum error is above 50% in some configurations for both WCAs. This result was obtained because the analysis assumes that the worst case always happens. Moreover, it considers that the traffic flows do not experience the worst

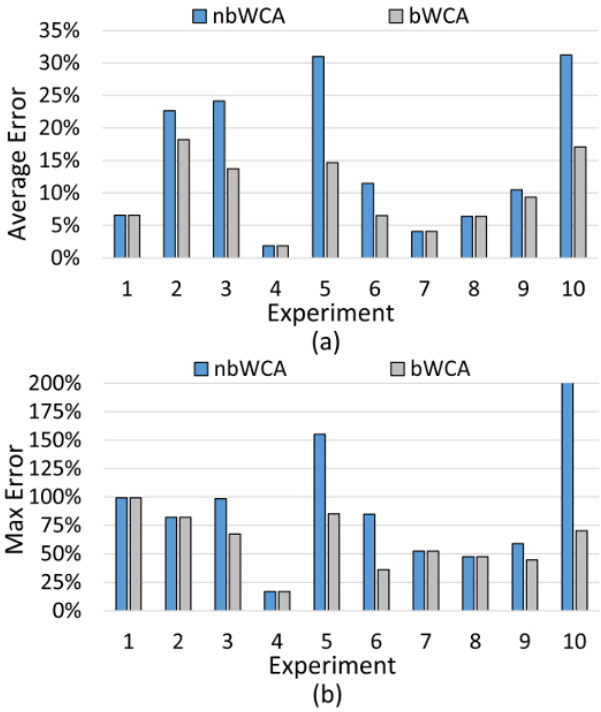


FIGURE 9: WCAs (a) average error and (b) maximum error

TABLE 3: nbWCA and bWCA execution time

Experiment	Indirect	Blocking		Time Increase (%)
		nbWCA	bWCA	
1	1	42	66	57.1
2	17	73	162	121.9
3	18	71	132	85.9
4	2	20	27	35
5	23	116	204	75.8
6	4	88	93	5.6
7	1	46	58	26.1
8	3	69	71	2.9
9	6	94	172	82.9
10	16	103	205	99.03

case in that simulation. However, bWCA correctly detected the whole network schedulability. On the other hand, nbWCA is too pessimistic and could not detect the schedulability in experiments 5 and 10.

As Table 3 shows, the execution time of the bWCA model is higher than nbWCA in all the experiments. In the bWCA model, every time an indirect blocking is identified, it is necessary to check its influence using Property 2, increasing the execution time. For this reason, experiments with more indirect blocking (influent or not) present a higher execution time. This time increase is acceptable because it remains in milliseconds. bWCA is four orders of magnitude faster than the simulation, with an average execution time of 119 ms, while the average simulation time is 2 hours.

B. SELF-BLOCKING ANALYSIS EVALUATION

We carried out nine simulations varying the depth of the buffers from 4 to 1024 flits using a simple scenario to

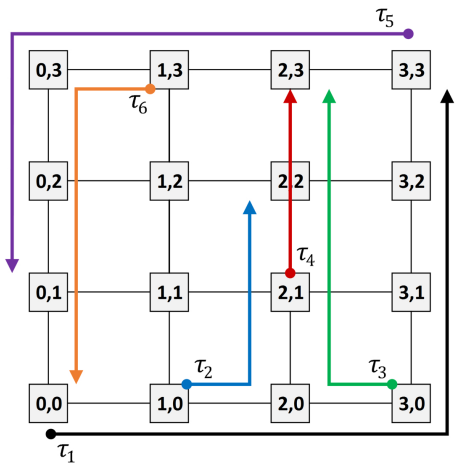


FIGURE 10: Scenario to evaluate the self-blocking analysis

TABLE 4: Traffic flows configuration to evaluate the self-blocking analysis.

τ_i	Source (x,y)	Destination (x,y)	Payload (flits)	Period (cycles)	Deadline (cycles)	VC
1	0,0	3,3	40	130	390	1
2	1,0	2,2	100	350	1050	1
3	3,0	2,3	60	165	495	0
4	2,1	2,3	40	190	570	0
5	3,3	0,1	40	130	390	1
6	1,3	0,0	300	550	1650	0

evaluate the self-blocking analysis and the flow saturation. Fig. 10 depicts this scenario. We mapped six traffic flows, injecting 1000 packets into the NoC. Table 4 shows the configuration of each of these flows. The deadline is set to three times its period, allowing multiple packet releases and self-blocking. We set τ_1, τ_2, and τ_5 to use the lowest priority virtual channel to observe the self-blocking in those traffic flows. This configuration allows us to evaluate the self-blocking when several indirect blocking occurs due to traffic flows with small packets (in the case of τ_1 and τ_2). These flows can direct block each other and are indirectly blocked by τ_3 and τ_4. In the case of τ_5, only a direct blocking by τ_6 occurs. However, in this situation, we evaluate the direct blocking by a traffic flow with a more extensive packet during the self-blocking.

Table 5 presents the results of the self-blocking analysis. We omitted the results for buffers deeper than 128 flits because the maximum latency remains the same. The analysis presents a tighter estimative in configurations with 4-flit and 8-flit buffers (the average error is about 2%). The reason is that τ_1 reaches its saturation, and since the analysis can detect it, we assume 0% error to the estimative of this traffic flow. However, we discarded the result for the configuration with 16-flit buffers because the analysis detected a saturation that did not occur. In the other configurations, the maximum error increased up to 77%. The experiments demonstrated that the detection of flow saturation is very pessimistic when indirect blocking occurs. None of the other traffic flows saturated, and the analysis presented accurate results, reducing the average error to below 15%.

TABLE 5: Average and maximum error of self-blocking analysis.

Buffers size(flits)	Error margin (%)	
	Average	Maximum
4	1,83	10,1
8	2,00	11,1
16*	-	-
32	13,70	73
64	14,40	77,2
128	1,83	58,9

* The analysis detected an inexistent flow saturation

In this study, the performance of the proposed WCA models for scheduling analysis in NoCs was evaluated and compared with a cycle-accurate SystemC-based NoC simulator. The results show that the bWCA model, which considers the size of the buffers in the NoC, presents a lower average error and a tighter maximum latency compared to the nbWCA model, which does not consider the size of the buffers. This improvement is mainly due to the correct detection of indirect blocking influence in the bWCA model, which reduces pessimism. The execution time of the bWCA model is higher than the nbWCA model, but it remains in milliseconds and is four orders of magnitude faster than the simulation.

For comparison purposes, Table 6 presents a summary of some related works by identifying their features, which features of the present work they did not consider, and the error range they obtained (when reported).

VII. CONCLUSIONS AND FUTURE WORK

This paper presented the schedulability analysis of a wormhole switching NoC with XY routing and round-robin arbiters. In the proposed worst-case analysis, we assumed a set of periodic tasks with a deadline less or equal to its period. The analysis also considered the buffer size of each router to determine the indirect blocking influence. In addition, we presented an extension of the analysis, including self-blocking and assuming tasks with a deadline greater than its period.

To evaluate the proposed WCA, we conducted a series of experiments using a cycle-accurate SystemC-based NoC simulator and compared its results with the ones of the WCA. In order to evaluate the pessimism reduction, we also used a version of the WCA without considering the size of the buffers. The self-blocking analysis was evaluated using a simple scenario and varying the size of the buffers.

The experiments demonstrated that the proposed WCA considering the size of the buffers provides a tighter estimative in configurations where indirect blocking occurs in higher quantities. A significant reduction in the average and maximum error can also be observed. On the other hand, the self-blocking analysis provided a tighter estimate for traffic flows that suffer only direct blocking.

In addition, the experimental results demonstrated that the proposed WCA could be used in design flows for NoC-based

TABLE 6: Results from literature-related works.

Reference	Metric	Main features and error in relation to simulation
[15]	maximum latency	static arbitration, did not consider buffers in the model, and did not present the average error in the text.
[17]	maximum latency	did not consider the depth of buffers in the model and did not present the error in relation to simulation, only the improvement of the analysis compared to a previous work
[18]	worst-case inter-core communication latency	static arbitration, only evaluated in the direct blocking model and did not consider the buffer size, error less than 10%
[20]	worst-case end-to-end latencies	simulated a specific scenario with only three flows, did not consider buffer size, static arbitration, analysis error varied from 32% to 11%
[21]	maximum latency	did not consider the buffer size and did not reported the error because the simulator was not able to reach the worst case
[22]	maximum latency	only made the comparison of routing algorithms and did not show the error in relation to simulation
[23]	maximum latency	reduced the error by 11.38% for packets with 2 flits and by 1.27% for packets with 19 flits

real-time systems to evaluate the schedulability of tasks with temporal requirements. The WCA can provide a preliminary evaluation of task mappings, avoiding unnecessary simulations of non-schedulable configurations and, consequently, a reduction in the exploration time of the design space.

In future work, we intend to use the proposed WCA as the fitness function in a task mapping design flow, as mentioned before, to accelerate the design space exploration. Also, we intend to improve the self-blocking analysis to reduce its pessimism. Finally, we may adapt the proposed analysis to an NoC with more than two virtual channels and different communication mechanisms.

REFERENCES

[1] P. Guerrier and A. Greiner, "A generic architecture for on-chip packetswitched interconnections," in Conference on Design, Automation and Test in Europe, DATE '00, (Paris, France), p. 250–256, March 2000.

[2] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist, "Network on a chip: An architecture for billion transistor era," in Norchip Conference, (Turku, Finland), p. 166–173, November 2000.

[3] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in 38th Annual Design Automation Conference, DAC '01, (Las Vegas, USA), p. 684–689, June 2001.

[4] F. Karim, A. Nguyen, S. Dey, and R. Rao, "On-chip communication architecture for oc-768 network processors," in 38th Design Automation Conference, DAC '01, (Las Vegas, USA), p. 678–683, June 2001.

[5] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," IEEE Computer, vol. 35, no. 1, p. 70–78, 2002.

[6] W. Chen and et al., "Evolution of publications, subjects, and co-authorships in network-on-chip research from a complex network perspective," IEEE Access, vol. 9, pp. 149399–149422, 2021.

[7] M. F. Reza, "Deep reinforcement learning enabled self-configurable networks-on-chip for high-performance and energy-efficient computing systems," IEEE Access, vol. 10, pp. 65339–65354, 2022.

[8] B. B. Yusuf, T. Maqsood, F. Rehman, and S. A. Madani, "Energy aware parallel scheduling techniques for network-on-chip based systems," IEEE Access, vol. 9, pp. 38778–38791, 2021.

[9] A. Sarihi, A. Patooghy, A. Khalid, M. Hasanzadeh, M. Said, and A. H. A. Badawy, "A survey on the security of wired, wireless, and 3d network-on-chips," IEEE Access, vol. 9, pp. 107625–107656, 2021.

[10] A. Burns and A. Wellings, Real-time systems and programming languages. Canada: Pearson Education, fourth ed., 2009.

[11] A. Khare, I. A. Boben, and S. Chattopadhayay, "Enhanced schedulability via minimal routing with mapping and priority assignment for real-time network-on-chip," in TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), pp. 564–568, 2019.

[12] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in Second ACM/IEEE International Symposium on Networks-on-Chip, NOCS '08, (Newcastle upon Tyne, UK), p. 161–170, April 2008.

[13] P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," IEEE Transactions on Computers, vol. 54, no. 8, p. 1025–1040, 2005.

[14] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in Real-Time Systems Symposium, (Lake Buena Vista, USA), p. 201–209, December 1990.

[15] Z. Shi, A. Burns, and L. Indrusiak, "Schedulability analysis for real time on-chip communication with wormhole switching," International Journal of Embedded and Real-Time Communication Systems, vol. 1, no. 2, p. 1–22, 2010.

[16] L. Indrusiak, "End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration," Journal of Systems Architecture, vol. 60, no. 7, p. 553–561, 2014.

[17] M. Liu, M. Becker, M. Behnam, and T. Nolte, "Tighter time analysis for real-time traffic in on-chip networks with shared priorities," p. 1–8, August 2016.

[18] Y. Ding and W. Zhang, "Hop-based priority scheduling to improve worst-case inter-core communication latency," in 12th IEEE International Conference on Embedded and Ubiquitous Computing, EUC '14, (Milano, Italy), p. 52–57, August 2014.

[19] B. Nikolić and S. Petters, "Edf as an arbitration policy for wormholeswitched priority-preemptive nocs," in 14th International Conference on Embedded Software, EMSOFT '14, (Jaypee Greens, India), p. 1–10, October 2014.

[20] J. Diemer, J. Rox, M. Negrean, S. Stein, and R. Ernst, "Real-time communication analysis for networks with two-stage arbitration," in International Conference on Embedded Software, EMSOFT '11, (Taipei, Taiwan), p. 243–252, October 2011.

[21] E. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in Conference on Design, Automation and Test in Europe, DATE '15, (Grenoble, France), p. 537–542, March 2015.

[22] M. Imai and T. Yoneda, "Performance modeling and analysis of on-chip networks for real-time applications," in 18th Pacific Rim International Symposium on Dependable Computing, PRDC '12, (Niigata, Japan), p. 111–120, November 2012.

[23] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "Wormhole networks properties and their use for optimizing worst case delay analysis of many-cores," in 10th IEEE International Symposium on Industrial Embedded Systems, pp. 1–10, IEEE, 2015.

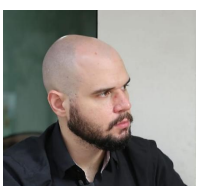
[24] S. Dahule, R. Golhar, and M. Ramteke, "The behavior of round robin arbiter in noc architecture," International Journal of Engineering and Innovative Technology, vol. 3, no. 5, pp. 312–314, 2013.

[25] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," Software Engineering Journal, vol. 8, no. 5, pp. 284–292, 1993.

[26] E. da Silva, M. Kreutz, and C. Zeferino, "Redscarf: an open-source multi-platform simulation environment for performance evaluation of networks-on-chip," Journal of Systems Architecture, vol. 99, p. 101633, 2019.

RUBENS VICENTE DE LIZ BOMER has a bachelor's degree in Computer Science from the University of Vale do Itajaí (Univali), Brazil, in 2014 and a master's degree in Applied Computer Science from Univali, in 2017. Mr. Bomer also has experience in the field of Computer Science, with a focus on Embedded Systems. He was a member of the Laboratory of Embedded and Distributed Systems (LEDS) at Univali from 2011 to 2017.

CESAR ALBENES ZEFERINO (Member, IEEE) received his Ph.D. in Computer Science from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2003, with a sandwich internship at the Sorbonne University, Paris, France. He is a Full Professor of the School of Sea, Science and Technology (EMCT) of the University of Vale do Itajaí (Univali), Brazil, since 2002, teaching undergraduate and graduate courses in the fields of Computer Architecture, Embedded Systems, and Digital Systems. Currently, he is the Director of EMCT and the head of the Laboratory of Embedded and Distributed Systems (LEDS). He is granted a researcher productivity scholarship from the National Council for Scientific and Technological Development – CNPq, Brazil. His interests include Digital Systems Design, Embedded Systems Design, Networks-on-Chip, Hardware Accelerators, and the Internet of Things.



LAIO ORIEL SELMAN received his bachelor's (2013) and Master's (2015) degrees in Electrical Engineering from the Regional University of Blumenau (FURB) and Ph.D. degree (2017) in Electrical Engineering from the Federal University of Santa Catarina (UFSC). He is currently a permanent professor at the Master Program in Applied Computer Science at the University of Vale do Itajaí (Univali) and a collaborating professor at the Postgraduate Program in Automation and Systems at UFSC. He has experience in Electrical Engineering with an emphasis on computational systems, working mainly on modeling, control, and optimization strategies (linear, non-linear, and mixed integer programming) and applied artificial intelligence. His main applications include intelligent transport systems, cyber-physical systems, aerospace systems (CubeSats), and oil and gas production.

VALDERI REIS QUIETINHO LEITHARDT (Member, IEEE) received the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2015. He is currently an Adjunct Professor with the Polytechnic Institute of Portalegre and a Researcher integrated with the VALORIZA Research Group, School of Technology and Management (ESTG). He is also a Collaborating Researcher with the following research groups, such as COPELABS, Universidade Lusófona de Lisboa, Portugal, and the Expert Systems and Applications Laboratory, University of Salamanca, Spain. His research interests include distributed systems with a focus on data privacy, communication, and programming protocols, involving scenarios and applications for the Internet of Things, smart cities, big data, cloud computing, and blockchain.