Buffer-Aware Worst-Case Timing Analysis of Wormhole NoCs Using Network Calculus

Frédéric Giroudot, Ahlem Mifdaoui

Buffer-Aware Worst-Case Timing Analysis of Wormhole NoCs Using Network Calculus

Frédéric Giroudot
Complex Systems Engineering Dept
ISAE – Université de Toulouse
Toulouse, France
frederic.giroudot@isae.fr

Ahlem Mifdaoui
Complex Systems Engineering Dept
ISAE – Université de Toulouse
Toulouse, France
ahlem.mifdaoui@isae.fr

***Abstract*—Conducting worst-case timing analyses for wormhole Networks-on-chip (NoCs) is a fundamental aspect to guarantee real-time requirements, but it is known to be a challenging issue due to complex congestion patterns that can occur. In that respect, we introduce in this paper a new *buffer-aware timing analysis* of wormhole NoCs based on Network Calculus. Our main idea consists in considering the flows serialization phenomena along the path of a flow of interest (f.o.i), by paying the bursts of interfering flows only at the first convergence point, and refining the interference patterns for the f.o.i accounting for the limited buffer size. Moreover, we aim to handle such an issue for wormhole NoCs, implementing a fixed priority-preemptive arbitration of Virtual Channels (VCs), that can be assigned to an arbitrary number of traffic classes with different priority levels, *i.e. VC sharing*, and each traffic class may contain an arbitrary number of flows, *i.e. priority sharing*. It is worth noting that such characteristics cover a large panel of wormhole NoCs.**

The derived delay bounds are analyzed and compared to available results of existing approaches, based on Scheduling Theory as well as Compositional Performance Analysis (CPA). In doing this, we highlight a noticeable enhancement of the delay bounds tightness in comparison to CPA approach, and the inherent safe bounds of our proposal in comparison to Scheduling Theory approaches. Finally, we perform experiments on a manycore platform, to confront our timing analysis predictions to experimental data and assess its tightness.

***Index Terms*—Networks-on-chip, Network Calculus, real time, worst-case timing analysis, wormhole routing, priority sharing, VC-sharing, backpressure, flows serialization.**

I. Introduction

Networks-on-chip (NoC) have become the standard interconnect for manycore architectures because of their high throughput and low latency capabilities. Most NoCs use wormhole routing [1] to transmit packets over the network: the packet is split in constant length words called *flits*. Each flit is then forwarded from router to router, without having to wait for the remaining flits. Compared to store and forward (S&F) mechanisms, the wormhole routing allows to drastically reduce the storage buffers at each router, as well as the contention-free end-to-end latency of a packet, *i.e.* almost insensitive to the packet path length. On the other hand, the wormhole routing complicates the possible congestion patterns, since a packet waiting for a resource to be freed can occupy several input buffers of routers along its path; thus introducing indirect blocking delays due to the *buffer backpressure*[1] [2].

Hence, an appropriate timing analysis taking into account these phenomena has to be considered, to provide safe latency bounds in wormhole NoCs.

Various timing analysis approaches of such NoCs have been proposed in the literature, and the most relevant ones can broadly be categorized under three main classes: Scheduling Theory-based ([2]–[5]), Compositional Performance Analysis (CPA)-based ([6], [7]) and Network Calculus-based ([8]–[10]). However, these existing approaches suffer from some limitations making them applicable under very specific assumptions, and/or leading to pessimistic bounds on end-to-end latencies. These limitations are mainly due to:

- *considering specific assumptions*, such as: (i) distinct priorities and unique virtual channel assignment for each traffic flow in a router [3] [2]; (ii) a priority-share policy, but with a number of Virtual Channels (VC) at least equal to the number of traffic priority levels like in [4] [5] [8] [6] or the maximum number of contentions along the NoC [11];
- *ignoring the buffer backpressure phenomena*, such as in [7] [9] [10];
- *ignoring the flows serialization phenomena*[2] *along the flow path* by conducting an iterative response time computation, *i.e.* commonly used in Scheduling Theory and CPA, which generally leads to pessimistic delay bounds;
- *ignoring the buffer size impact on the interference patterns* through commonly considering that a packet occupies its complete path during its maximum end-to-end latency.

Hence, our main objective in this paper is coping with these limitations:

- First, we introduce a new *Buffer-aware timing analysis* of wormhole NoCs based on Network Calculus [12], and particularly the *Pay Multiplexing Only Once (PMOO)* principle [13], to enhance the end-to-end delay bounds

[1] A logical mechanism to control the flow on a communication channel and avoid buffer overflow.

[2] The pipelined behavior of networks infers that the interference between flows along their shared subpaths should be counted only once, i.e., at their first convergence point.

accuracy of such networks. Our main idea consists in considering: (i) the flows serialization phenomena along the path of a *flow of interest (f.o.i)*, by paying the bursts of interfering flows only at the first convergence point; (ii) refined interference patterns for the f.o.i accounting for the limited buffer size, through quantifying the way a packet can spread on a NoC with small buffers under contentions. Moreover, we aim to handle such an issue for wormhole NoCs, implementing a fixed priority-preemptive arbitration of VCs, that can be assigned to an arbitrary number of traffic classes with different priority levels, *i.e. VC sharing*, and each traffic class may contain an arbitrary number of flows, *i.e. priority sharing*. Such characteristics cover a large panel of wormhole NoC routers. These results constitute our main theoretical contributions;

- Second, we analyse the derived delay bounds and compare them to available results of existing approaches, based on Scheduling Theory [3] [2] as well as CPA [6]. In doing this, we highlight a noticeable enhancement of the delay bounds in comparison to CPA approach, and the inherent safe bounds of our proposal in comparison to Scheduling Theory approaches. Finally, we perform experiments on a manycore platform, to confront our timing analysis predictions to experimental data and assess the analytical delay bounds tightness. To the best of our knowledge, the latter aspect is rarely handled in the literature, since the most relevant works in NoC timing analysis rely only on simulation to evaluate their analytical model. These results constitute our main practical contributions.

The remainder of this paper is organized as follows : we first review the main related work in Section II. Afterwards, we recall the main Network Calculus concepts, and present the system model and an overview of the main steps of our proposed analysis methodology in Section III. Then, the introduced refined interference patterns and the delay bound computation, taking into account the buffer size and the flows serialization impacts, are detailed in Sections IV and V, respectively. In addition, we report the comparative analysis of recent works in the same field against ours, and the derived experimental results in Section VI. Finally, we draw the main conclusions and future work in Section VII.

II. Related Work

The most relevant timing analyses of NoCs are mainly using two approaches: Scheduling Theory and Network Calculus, to which we can add an "hybrid" method known as Compositional Performance Analysis (CPA). The main supported assumptions of each one of these analyses are summarized in Table I, and we particularly consider: (i) the use of *wormhole routing* with *multiple VCs*; (ii) the support of *VCs Sharing* (*i.e.* many traffic classes per VC) and *priority Sharing* (*i.e.* many traffic flows per priority level) in each router; (iii) the integration of the *buffer size* and the flows *serialization* impacts.

Scheduling Theory has been used in [3] to derive worst-case latency bounds in wormhole NoCs supporting multiple VCs and with no priority sharing. Then, this method has been extended in [4] to support the priority sharing. However, the latter approach may lead to overly pessimistic results for large NoCs with a high number of flows and a limited number of virtual channels. Moreover, the authors in [2] have proved later that the method in [3] could be optimistic in specific situations, and refined the response time analysis through the distinction between downstream and upstream indirect interferences, which leads to a deeper understanding of the problem. However, they focus only on configurations with no priority sharing, which limits the applicability of such a proposal. Interesting enhancements of the work in [4] have been detailed in [5], where refined interference patterns between flows have been proposed through accounting for the physical contention domain impact, but considering only one-flit size buffers.

More recently, the authors in [7] have developed a model based on CPA, supporting priority sharing and VC sharing but ignoring the buffer backpressure. Afterwards, they have extended this work to support the backpressure in [6]. However, the proposed analysis has considered only a single VC with buffer sizes do not go below one packet, and ignored the flows serialization phenomena.

Finally, there are some recent work based on Network Calculus [12]. In [8], the authors have solved the chain blocking problem due to the wormhole routing in a recursive way, through modeling the flow control system as a router component, whose behavior depends on the following router. This method infers complex fixed-point problems solving and has been validated only on relatively small NoCs with a simple flows configuration; thus limiting their approach applicability and scalability. Moreover, they have considered only a single-VC NoC routers. Afterwards, the authors in [9] have provided tighter delay bounds, using improved arrival and service curves while taking buffer size and flows serialization into account. However, their approach seeks to provide a buffer size threshold to avoid the buffer backpressure. In addition, they do not consider wormhole routing as the switching technique; thus avoiding the complex chain blocking issue. Finally, the authors in [10] have started the exploration of the buffer size impact on the interference patterns. However, the considered wormhole routers do not support the VC concept and the proposed approach does not integrate the flows serialization phenomena.

As illustrated in Table I, there is no existing timing analysis approach, which covers all the technical characteristics of wormhole NoC routers and supports the buffer size and flows serialization phenomena. Hence, our main contribution in this paper is to cope with these different characteristics and assumptions to compute accurate end-to-end delay bounds in wormhole NoCs, using Network Calculus.

Approach	Contribution	wormhole	multiple VCs	priority sharing	VCs sharing	flows serialization	buffer size B (*vs* packet length L) 1 flit	$L \leq B$	$B \leq L$
Scheduling Theory	[2]	x	x				x	x	x
	[4]	x	x	x					
	[5]	x	x	x			x		
CPA	[6]	x	-	x				x	
	[7]	x	x	x	x			x	
Network Calculus	[8]	x		x			x	x	x
	[9]		x	x		x		x	
	our approach	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**

TABLE I
SUMMARY OF THE ASSUMPTIONS OF THE MAIN EXISTING APPROACHES

III. TIMING ANALYSIS METHODOLOGY

To conduct our proposed timing analysis of wormhole NoCs using Network Calculus, we first present the main concepts of Network Calculus that will be used in the paper. Afterwards, we detail the system assumptions and model. Finally, we give an overview of the main followed steps to compute the end-to-end delay bounds. The main notations used in this paper are in Table II, where upper indices indicate a node or a set of nodes and lower indices indicate flows.

A. Network Calculus Background

Network Calculus describes data flows by means of cumulative functions, defined as the number of transmitted bits during the time interval $[0, t]$. Consider a system S receiving input data flow with a Cumulative Arrival Function (CAF), $A(t)$, and putting out the same data flow with a Cumulative Departure Function (CDF), $D(t)$. To compute upper bounds on the worst-case delay and backlog, we need to introduce the maximum arrival curve, which provides an upper bound on the number of events, e.g., bits or packets, observed during any interval of time.

Definition 1. *(Arrival Curve) [12] A function α is an arrival curve for a data flow with the CAF A, iff:*

$$\forall t, s \geq 0, s \leq t, A(t) - A(s) \leq \alpha(t - s)$$

A widely used curve is the leaky bucket curve, which guarantees a maximum burst σ and a maximum rate ρ, *i.e.*, the traffic flow is (σ, ρ)-constrained. In this case, the arrival curve is defined as $\gamma_{\sigma,\rho}(t) = \sigma + \rho \cdot t$ for $t > 0$. Furthermore, we need to guarantee a minimum offered service within crossed nodes through the concept of minimum service curve.

Definition 2. *(Simple Minimum Service Curve) [12] The function β is the simple service curve for a data flow with the CAF A and the CDF D, iff:*

$$\forall t \geq 0, D(t) \geq \inf_{s \leq t}(A(s) + \beta(t - s))$$

A very useful and common model of service curve is the rate-latency curve $\beta_{R,T}$, with R the minimum guaranteed rate and T the maximum latency before starting the service. This rate-latency function is defined as $\beta_{R,T}(t) = [R \cdot (t - T)]^+$, where $[x]^+$ is the maximum between x and 0. Knowing the arrival and service curves, one can compute the upper bounds on performance metrics for a data flow, according to the following theorem.

Theorem 1. *(Performance Bounds) Consider a flow constrained by an arrival curve α crossing a system $\mathcal{S}$ that offers a service curve β, then:*
Delay [3]*:* $\forall\, t:\ d(t) \leq h(\alpha, \beta)$
Backlog [4]*:* $\forall\, t:\ q(t) \leq v(\alpha, \beta)$
Output arrival curve [5]*:* $\alpha^*(t) = \alpha \oslash \beta(t)$

The calculus of these bounds is greatly simplified in the case of a leaky bucket arrival curve and a rate-latency service curve. In this case, the delay and backlog are bounded by $\frac{\sigma}{R} + T$ and $\sigma + \rho \cdot T$, respectively; and the output arrival curve is $\sigma + \rho \cdot (T + t)$.

Finally, we will extend the following results concerning the end-to-end service curve of a f.o.i. accounting for flows serialization effects in feed-forward networks, based on the PMOO principle [13], under non-preemptive Fixed Priority (FP) multiplexing.

Theorem 2. *The service curve offered to a flow of interest f along its path $\mathbb{P}_f$, in a network under non-preemptive FP multiplexing with strict service curve nodes of the rate-latency type $\beta_{R,T}$ and leaky bucket constrained arrival curves $\alpha_{\sigma,\rho}$, is a rate-latency curve, with a rate $R^{\mathbb{P}_f}$ and a latency $T^{\mathbb{P}_f}$, as follows :*

$$R^{\mathbb{P}_f} = \min_{k \in \mathbb{P}_f} [R^k - \sum_{i \ni k, i \in shp(f)} \rho_i] \tag{1a}$$

$$T^{\mathbb{P}_f} = \sum_{k \in \mathbb{P}_f} \left(T^k + \frac{\max\limits_{i \ni k, i \in slp(f)} L_i}{R^k} \right) + \sum_{i \in DB_f \cap shp(f)} \frac{\sigma_i^{cv(i,f)} + \rho_i \cdot \sum\limits_{k \in \mathbb{P}_f \cap \mathbb{P}_i} \left(T^k + \frac{\max\limits_{i \ni k, i \in slp(f)} L_i}{R^k} \right)}{R^{\mathbb{P}_f}} \tag{1b}$$

, where $slp(f)$ (resp. $shp(f)$) denotes all flows with a priority lower (resp. higher) or equal than f and L_f the maximal packet size of flow f.

[3] $h(f, g)$: the maximum horizontal distance between f and g
[4] $v(f, g)$: the maximum vertical distance between f and g
[5] $f \oslash g(t) = \sup_{\forall u \geq 0}\{f(t + u) - g(u)\}$

Notation	Definition
$\mathcal{F}$	the set of flows on the NoC
S_{flit}	The size of one flit
B	The buffer size of a VC in a router
$\mathbb{P}_f$	The list of nodes crossed by f from source to destination
$\mathbb{P}_f[k]$	The $k+1^{th}$ node of f path
$subpath(l,k)$	The subpath of flow l relatively to flow k just after dv(k,l)
$Last(l,k)$	The index of dv(k,l) in $\mathbb{P}_l$
cv(i,j)	The convergence node of i and j
dv(i,j)	The divergence node of i and j
$f \ni r$	Flow f crosses node r
$F \supset r$	There is a flow $f \in F$ such that $f \ni r$
$\mathbf{1}_{\{cdt\}}$	equals 1 if cdt is true and zero otherwise
L_f	The maximal packet length of f
J_f	The release jitter of f
P_f	The period of f
$\alpha_f(t)$	The initial arrival curve of f
$\alpha_f^r(t)$	The arrival curve of f at node r
σ_f^r	The burst of α_f^r
ρ_f^r	The rate of α_f^r
$\beta_f(t)$	The end-to-end service curve of f
$\beta_f^{\mathbb{P}_f[i:j]}(t)$	The service curve granted to f on the path from $\mathbb{P}_f[i]$ to $\mathbb{P}_f[j]$
$R_f^{\mathbb{P}_f[i:j]}$	The rate of $\beta_f^{\mathbb{P}_f[i:j]}$
$T_f^{\mathbb{P}_f[i:j]}$	The latency of $\beta_f^{\mathbb{P}_f[i:j]}$
$\widetilde{\beta}_k^{subP}(t)$	The VC-service curve of f on $subP$
$\widetilde{R}_f^{subP}$	The rate of $\widetilde{\beta}_k^{subP}$
$\widetilde{T}_f^{subP}$	The latency of $\widetilde{\beta}_k^{subP}$
DB_f	set of all flows directly interfering with f
$DB_f^{\|path}$	Flows $i \in DB_f$ such that $\mathbb{P}_i \cap path \neq \varnothing$
$hp(f)$	Flows mapped to a VC of strict higher priority than f
$sp(f)$	Flows mapped to the same VC as f
$lp(f)$	Flows mapped to a VC of strict lower priority than f
$slp(f)$	All flows with a priority lower or equal than f, f excluded
$shp(f)$	All flows with a priority higher or equal than f, f excluded
IB_f	Indirect blocking set of flow f
N_f	The spread index for f, i.e., number of buffers to store f
$D_f^{\mathbb{P}_f}$	The end-to-end delay bound of f

TABLE II
SUMMARY OF NOTATIONS

B. System Model And Assumptions

Our model can apply to an arbitrary NoC topology as long as the flows are routed in a deterministic, deadlock-free way (see [1]). Nonetheless, we consider the commonly used 2D-mesh topology with input-buffered routers and XY-routing, due to their simplicity and high scalability. The considered wormhole NoC routers, illustrated in Fig. 1 (left), implement a priority-based arbitration of VCs and enable flit-level preemption through VCs. The latter can happen if a flow from a higher priority VC asks for an output that is being used by the f.o.i.. Hence, when the flit being transmitted finishes its transmission, the higher priority flow is granted the use of the output while the f.o.i. waits. Moreover, each VC has a specific input buffer and supports many traffic classes, i.e., VCs sharing, and many traffic flows may be mapped on the same priority-level, i.e., priority sharing. Finally, the implemented VCs enable the bypass mechanism, illustrated in Fig. 2. If the f.o.i gets blocked at some point (for instance, flow 1 in Fig. 2), flows from lower priority VCs sharing upstream outputs with the f.o.i (for instance, flow 2 in Fig. 2) can bypass it, but they will be preempted again when the downstream blocking of the f.o.i. disappears.

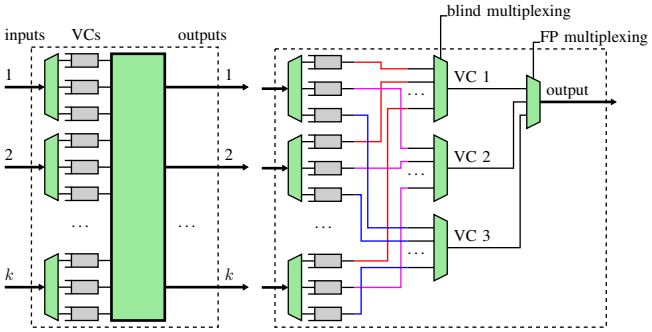

Fig. 1. Architecture of an input-buffered router (left) and output multiplexing (right) with the arbitration modeling choices

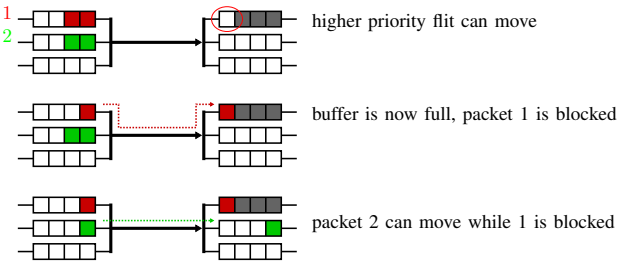

Fig. 2. Bypass mechanism

We consider an arbitrary service policy to serve flows belonging to the same VC within the router, *i.e.*, these flows can be from the same traffic class or from different traffic classes mapped on the same VC. This assumption allows us to cover the worst-case behaviors of different service policies, such as FIFO and Round Robin (RR) policies.

Hence, we model such a wormhole NoC router as a set of independent hierarchical multiplexers, where each one represents an output port as shown in Fig. 1 (right). The first arbitration level is based on a blind (arbitrary) service policy to serve all the flows mapped on the same VC level and coming from different *geographical* inputs; whereas the second level implements a preemptive Fixed Priority (FP) policy to serve the flows mapped on different VCs levels and going out from the same output port. It is worth noticing that the independency of the different output ports is guaranteed in our model, due to the integration of the flows serialization phenomena. The latter infers ignoring the interference between the flows entering a router through the same input and exiting through different outputs, since these flows have necessarily arrived through the same output of the previous router, where we have already taken into account their interference.

Each router-output pair r (that we will refer to as a *node* from now on) has a processing capacity that we model using

a rate-latency service curve.

$$\beta^r(t) = R^r(t - T^r)^+$$

R^r represents the minimal processing rate of the router for this output (which is typically expressed in flits per cycle) and T^r the maximal experienced delay by any flit crossing the router before being processed (which is commonly called routing delay and takes one or few cycles).

On the other hand, the characteristics of each traffic flow $f \in \mathcal{F}$ are modeled with the following leaky bucket arrival curve, which covers a lot of different traffic arrival events, such as periodic and sporadic with or without jitter.

$$\alpha_f(t) = \sigma_f + \rho_f \cdot t$$

This arrival curve integrates the maximal packet length L_f (payload and header in flits), the period or minimal inter-arrival time P_f (in cycles), and the release jitter J_f (in cycles) in the following way :

$$\begin{aligned} \rho_f &= \frac{L_f}{P_f} \\ \sigma_f &= L_f + J_f \cdot \rho_f \end{aligned}$$

For each flow f, its path $\mathbb{P}_f$ is the list of nodes (router-outputs) crossed by f from source to destination. Moreover, for any k in appropriate range, $\mathbb{P}_f[k]$ denotes the $k+1^{th}$ node of flow f path (starting at index 0). Therefore, for any $r \in \mathbb{P}_f$, the propagated arrival curve of flow f from its initial source until the node r, computed based on Th. 1, will be denoted:

$$\alpha_f^r(t) = \sigma_f^r + \rho_f^r \cdot t$$

The end-to-end service curve granted to flow f on its whole path will be denoted:

$$\beta_f(t) = R_f\,(t - T_f\,)^+$$

And similarly, the service curve granted to f on a partial path from node $\mathbb{P}_f[i]$ to node $\mathbb{P}_f[j]$ is as follows:

$$\beta_f^{\mathbb{P}_f[i:j]}(t) = R_f^{\mathbb{P}_f[i:j]}\left(t - T_f^{\mathbb{P}_f[i:j]}\right)^+$$

C. Main Steps of the Delay Bounds Computation

To compute an upper bound on the end-to-end latency for the f.o.i f, we need to define its granted end-to-end service curve:

$$\beta_f(t) = R_f\,(t - T_f\,)^+$$

The rate R_f represents the bottleneck rate along the flow path, accounting for directly interfering flows of same and higher priority than f. The latency T_f consists of several parts :

$$T_f = T_{hp} + T_{sp} + T_{lp} + T_{IB} + T_{\mathbb{P}_f} \tag{2}$$

- $T_{\mathbb{P}_f}$ is the "base latency", that any flit of f experiences along its path due only to the technological latencies of the crossed routers;
- T_{hp} is part of the maximum direct blocking latency, due to flows of higher priority sharing resources with f, which are part of the *direct blocking* set DB_f;
- T_{sp} is part of the maximum direct blocking latency, due to same priority flows sharing resources with f, and also part of DB_f set;
- T_{lp} is part of the maximum direct blocking latency, due to flows of lower priority sharing resources with f, and also part of DB_f set;
- T_{IB} is the maximum indirect blocking latency, due to flows that can indirectly block f through the buffer backpressure phenomenon, and these flows are denoted *indirect blocking* set IB_f.

The *direct blocking* set DB_f and *indirect blocking* set IB_f will be detailed in Section IV. We illustrate symbolically these influences on Figure 3. The f.o.i undergoes direct blocking along its path, while indirect blocking occurs on "branches" and backpropagates to the f.o.i.

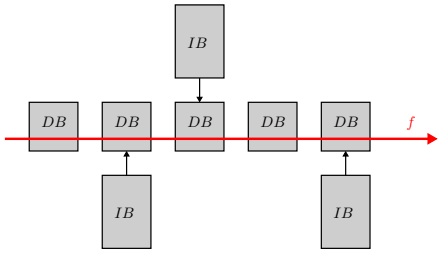

Fig. 3. Symbolic representation of contributions to latency T_f of flow f

Hence, our delay bounds computation for a f.o.i f is based on three main steps:

1) First, we will refine the indirect blocking set of flows, IB_f, which are involved in the indirect interference of the f.o.i f, as well as the section of their path on which they induce this blocking. This step will take into account the impact of the limited buffer size on the way a packet can spread on the NoC; thus on IB_f, and it will be detailed in Section IV;

2) Second, we will compute the end-to-end service curve of flow f taking direct blocking and indirect blocking delays into account. This step integrates the flows serialization effects using the PMOO principle [13], and it will be detailed in Section V. We will start by analyzing the impact of the direct blocking set, DB_f, on the end-to-end service curve, *i.e.* T_{hp}, T_{sp} and T_{lp}. Afterwards, we will compute the impact of the indirect blocking set computed in the first step on such a curve, i.e T_{IB};

3) Finally, once the end-to-end service curve for the f.o.i f is known, we use Theorem 1 to get an upper bound on the end-to-end delay of f, $D_f^{\mathbb{P}_f}$, as follows:

$$D_f^{\mathbb{P}_f} = \frac{\sigma^{\mathbb{P}_f[0]}}{R_f} + T_{hp} + T_{sp} + T_{lp} + T_{IB} + T_{\mathbb{P}_f} \tag{3}$$

IV. Buffer-Aware Analysis of Indirect Blocking Sets

Indirect blocking set consists of flows that do not physically share any resource with the f.o.i, but can delay it because they impact flows that directly impact the f.o.i. It is worth noticing that this definition is slightly different from the one used in the Scheduling Theory approaches [4] [5], where there is a distinction between the indirect blocking, i.e., due to same-priority flows blocking one another and eventually impacting the f.o.i, and indirect interference, i.e., due to higher priority flows. In our approach, we only consider flows belonging to the same class as the f.o.i to compute the indirect blocking set, since the impact of higher priority flows is already integrated in our model as follows:

- if a higher priority flow blocking our f.o.i gets blocked, the f.o.i can bypass it. In this case, we take into account the extra processing delay needed to allocate the shared resource to the f.o.i. On the other hand, the buffer backpressure will only propagate among flows from the same class, as illustrated in Fig. 2;
- the influence of higher priority flows on the same priority flows than the f.o.i, which are inducing the indirect blocking, is modeled through the granted end-to-end service curve of each one of these flows at the rate and latency levels, as explained in Section III-C

To better understand the impact of the buffer size on the packet spreading, and consequently the indirect blocking set, we consider the example of Figure 4 under the following assumptions: (i) each buffer can store only one flit; (ii) all flows have 3-flit-long packets; (iii) all flows are mapped to the same VC; (iv) the f.o.i is flow 1.

In the upper configuration, a packet of flow 3 occupies buffers (7, 8, 9). Consequently, a packet of flow 2, waiting for buffer 7, is blocked and occupies buffers (4, 5, 6). In turns, a packet of the f.o.i flow 1 is blocked, since it needs buffer 4 to move forward. Hence, flow 3 is included in the indirect blocking set of the f.o.i flow 1.

In the bottom configuration, a packet of flow 4 occupying buffers (12, 13, 14) leads to a direct blocking of a packet of flow 2, waiting in buffers (7, 8, 11). However, this packet of flow 2 is too far away from the last node flows 1 and 2 have in common (buffer 4). Hence, flow 4 is not part of the indirect blocking set of the f.o.i flow 1, even though there is an intermediate flow that shares resources with both flows 1 and 4.

It is worth noticing that the indirect blocking set analysis without taking into account the buffer size will include flows 3 and 4 in IB_1 (the indirect blocking set of the f.o.i flow 1). However, as illustrated in the Figure 4, there is actually only flow 3 in the indirect blocking set of the f.o.i flow 1, due to the impact of the buffer size on the packet spreading.

Hence, the limited buffer size reduces the section of the path on which a blocked packet can in its turn block another

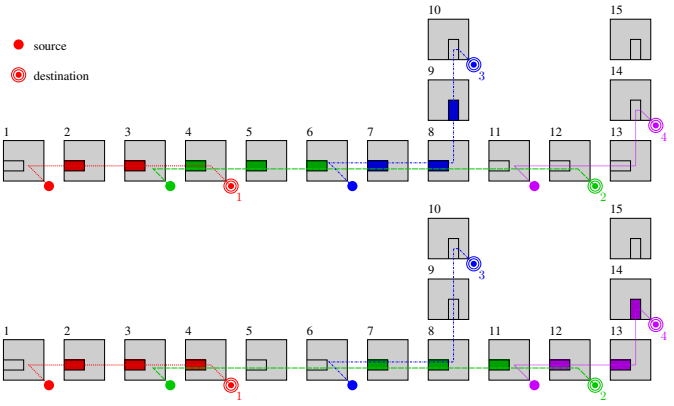

Fig. 4. Impact of packet size and buffer size on indirect blocking

one. Consider two flows k and l that are directly interfering with one another and let $\text{dv}(k,l)$ be the last node they share :

$$\text{dv}(k,l) = \mathbb{P}_k[\max\{i, l \ \ni \mathbb{P}_k[i]\}]$$

Suppose the path of l continues after this node. Even if the head flit of l is not stored in a router of $\mathbb{P}_k \cap \mathbb{P}_l$, the limited buffer size available in each router can lead to storing the tail flit of l in a router of $\mathbb{P}_k \cap \mathbb{P}_l$ under contention. In that case, l blocks k. Therefore, we need to quantify the way a packet of flow f spreads into the network when it is blocked and stored in buffers. We do so by computing N_f, the number of buffers needed to store one packet of flow f. We call N_f the "spread index for flow f". Given the maximum packet length of flow f, L_f flits, and the buffer size of a VC in routers B flits (supposed to be identical for all routers), N_f verifies :

$$N_f = \left\lceil \frac{L_f}{B} \right\rceil$$

Using the previous example, we call the section of the path of flow l from $\text{dv}(k,l)$ to N_l nodes (at most) after $\text{dv}(k,l)$ "subpath of flow l relatively to f.o.i k" :

$$subpath(l,k) = [\mathbb{P}_l[Last(l,k)+1], \ldots, \mathbb{P}_l[Last(l,k)+N_l]]$$

, where $Last(l,k) = \max\{n, \ \mathbb{P}_l[n] \in \mathbb{P}_k\}$ is the index of the last node shared by k and l along $\mathbb{P}_l$, i.e $\mathbb{P}_l[Last(l,k)] = \text{dv}(k,l)$. It is worth noticing that if $\mathbb{P}_l$ ends before reaching the N_l-th node after $\text{dv}(k,l)$, then we ignore the out-of-range indexes. The notion of subpath is illustrated in Fig. 5 for the f.o.i k and a spread index for the interfering flow l equal to 3, i.e., $N_l = 3$.

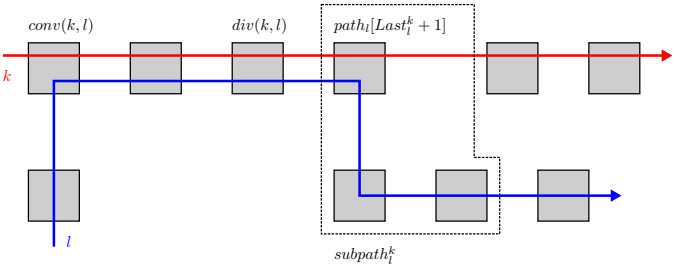

Fig. 5. Subpath illustration for the f.o.i k

The computation of the indirect blocking set IB_f is detailed in Algorithm 1.

To explain such an algorithm, we need the following definitions:

- $sp(f)$: all flows belonging to the same VC as f, f excluded ;
- $hp(f)$: all flows mapped to a VC of strict higher priority than the VC, to which f belongs;
- $lp(f)$: all flows mapped to a VC of strict lower priority than the VC, to which f belongs;
- DB_f: set of all flows directly interfering with flow f, regardless of their priority, i.e., $DB_f = \{i \in \mathcal{F},\ \mathbb{P}_i \cap \mathbb{P}_f \neq \varnothing\}$;
- $DB_f^{|path}$: set of flows of DB_f with paths having a non-null intersection with $path \subset \mathbb{P}_f$, *i.e.* $DB_f^{|path} = \{i \in DB_f,\ \ \mathbb{P}_i \cap path \neq \varnothing\}$.

The main steps of this algorithm when considering a f.o.i f are as follows:

1) determine all subpaths of flows in $DB_f^{|\mathbb{P}_f} \cap sp(f)$ relatively to flow f (lines 3-5);
2) for each of these subpaths, check if they intersect with other flows. If yes, determine the subpath of these other flows relatively to the subpath they intersect (lines 10-19);
3) reiterate until no new subpath is found (lines 7-21).

Notice that $DB_f \cap sp(f)$ and IB_f are disjoint sets, since the influence of directly-interfering, same-priority flows is already integrated through the computation of the direct blocking latency, which will be detailed in Section V-A. Moreover, Algorithm 1 can be used to compute the subset of IB_f containing only flows that can cause indirect blocking on a subpath $subP \subset \mathbb{P}_f$. In that case, we need to consider $DB_f^{|subP}$ instead of DB_f, and we will note the result "partial indirect blocking set" $IB_f^{|subP}$. We can also choose to ignore a subset of the flows in $\mathcal{F}$ during the algorithm (hence the optional parameter toIgnore). We can put in this subset the flows that we have already taken into account before. For instance, when we compute the impact of a flow i on the f.o.i f, we must exclude flow f from the flows impacting flow i. In fact, if not doing so, it would be equivalent to consider that flow f impacts itself, which is quite pessimistic.

The computational complexity of Algorithm 1, when considering a flow set $\mathcal{F}$ on the NoC, is denoted as $\mathcal{C}(|\mathcal{F}|)$ and is defined in the following property.

Property 1. *Consider a flow set $\mathcal{F}$, the computational complexity of Algorithm 1 is as follows:*

$$\mathcal{C}(|\mathcal{F}|) = \mathcal{O}\left(|\mathcal{F}| \cdot \left(K \cdot \max_{f \in \mathcal{F}} |\mathbb{P}_f|\right)\right) \tag{4}$$

, where K is a constant time to find the subpath of a flow relatively to the flow of interest,.

Proof. Consider f as the f.oi, we derive a computational complexity bound independent from f. We denote E_k the state of an arbitrary variable or set E after k iterations of the **while** loop (Line 7).

Algorithm 1 Determining IB_f and the associated subpaths

Input: f, the flow of interest, $\mathbb{P}_f$ the associated path, toIgnore a list of flows to ignore (optional)
Output: IB_f, a set containing flow indexes and associated subpath involved in indirect interference on $\mathbb{P}_f$

1: IB_f is initially empty
2: $init\_set \leftarrow DB_f^{|\mathbb{P}_f} \cap sp(f)$
   // *initialize S*
3: **for** $i \in init_set \backslash$toIgnore **do**
4: Append $\{i, i.\mathrm{subpath}(f)\}$ to S
5: **end for**
6: processedFlows $\leftarrow \varnothing$ // *Initialize processedFlows*
7: **while** $S \neq \varnothing$ **do**
8: Pop a pair $\{j, \mathrm{subj}\}$ from S
   // *Compute subpaths relatively to j on subj :*
9: currentDB $\leftarrow$ computeDBset(j, subj)
10: **for** (k, subk) in $(\mathrm{currentDB} \cap sp(j))$ **do**
11: **if** $k \in$ processedFlows $\cup\ init\_set \cup$ toIgnore **then**
12: Move directly to the next iteration of the loop
13: **end if**
14: **if** k has no entry in IB_f **then**
15: Add $\{k, \mathrm{subk}\}$ to IB_f and S
16: **else** // *there is an entry for k in IB_f*
17: Merge subk with the existing one in IB_f and S
18: **end if**
19: **end for**
20: Append j to processedFlows
21: **end while**
22: **return** IB_f

Moreover, for any flow $f_i \neq f$, k_{f_i} denotes the iteration at the end of which f_i is added to processedFlows for the first time, *i.e.* $k_{f_i} = \min\{k, f_i \in \mathrm{processedFlows}_k\}$, with the convention $k_{f_i} = \infty$ if f_i is never added to processedFlows. Let us show that f_i will be added to processedFlows at most once.

If $k_{f_i} < \infty$, there exists an iteration $k < k_{f_i}$ such that $f_i \in \mathrm{currentDB}_k$ and is added to S and IB_f for the first time (Line 15). If there exists $k_{f_i} > k' > k$ such that $f_i \in \mathrm{currentDB}_{k'}$, the corresponding subpath will be merged with the one already in S and IB_f (Line 17). This way, there will be only one entry for f_i in S and IB_f between iterations k and k_{f_i}. At iteration k_{f_i}, f_i is popped out of S and $\mathrm{currentDB}_{k_{f_i}} \subset DB_{f_i}$, thus does not contain f_i. Therefore, S does not contain f_i during the whole iteration. At the end of iteration k_{f_i}, processedFlows contains f_i. Consequently, from iteration $k_{f_i} + 1$ and forward, f_i cannot be added to S again (Line 11), thus neither to processedFlows. Hence, for any f_i, either $k_{f_i} = \infty$ or f_i is added only once to processedFlows.

Consider the sequence defined as $u_k = |\mathrm{processedFlows}_k|$. It is strictly increasing and upper-bounded by the number of flows in our configuration. This fact proves that there are at most $|\mathcal{F}|$ iterations of the **while** loop, after which S is necessarily empty. Hence the following holds :

$$\mathcal{C}(|\mathcal{F}|) = \mathcal{O}\left(|\mathcal{F}| \cdot \mathcal{C}(\mathrm{computeDBset})\right)$$

Let us now evaluate the complexity of computeDBset for a flow f.

Applied to a flow f, this function computes the subpaths of the flows in DB_f relatively to f. Assuming we have a preprocessed dictionary listing, for every node, the indexes of flows using this node,[6] we only have to run our algorithm through the path of f and check if there are contending flows at this node. Comparing the indexes of the current node with those of the previous node, we can find divergence nodes of contending flows relatively to the flow of interest. We assume that, knowing the divergence point of a contending flow relatively to the flow of interest, it takes a constant time K to find its subpath (we only need to compute the spread index). We finally have :

$$\mathcal{C}(|\mathcal{F}|) = \mathcal{O}\left(|\mathcal{F}| \cdot \left(K \cdot \max_{f \in F} |\mathbb{P}_f|\right)\right)$$

□

An application of Alg. 1 to compute IB_f is illustrated in Fig. 6, when considering the flow f as the f.o.i and four more flows with the same priority as f and a spread index of 3.

First, Algorithm 1 will initialize the set S with flow indexes and subpaths from DB_f :

$$S = \{\{1, S_1\}, \{2, S_2\}\}$$

Then, we enter the while loop :

1) we pop $\{1, S_1\}$ and see it does not intersect any other flow (line **??**, currentDB $= \varnothing$). We add 1 to processedFlows. S now contains $\{2, S_2\}$.
2) we pop $\{2, S_2\}$ and compute currentDB $= \{\ 3, S_3\}$. We add $\{3, S_3\}$ to IB_f and to S. We add 2 to processedFlows. S now contains $\{3, S_3\}$.
3) we pop $\{3, S_3\}$ and compute currentDB $= \{\ 4, S_4\}$. We add $\{4, S_4\}$ to IB_f and S We add 3 to processedFlows. S now contains $\{4, S_4\}$;
4) we pop $\{4, S_4\}$ and compute currentDB $= \varnothing$. We add 4 to processedFlows. S is now empty, the loop is over.

Thus we finally have :

$$IB_f = \{\{3, S_3\}, \{4, S_4\}\}$$

V. End-To-End Service Curves Computations

In this section, we will detail the second step of our computation methodology, which consists in defining the end-to-end service curve of each f.o.i taking into account the direct and indirect blocking impacts. First, we explain the computation of the direct blocking latency of the service curve, through extending the PMOO principle to wormhole NoCs. Then, we explain the computation of the indirect blocking latency of such a curve, based on the indirect blocking set defined in Section IV.

[6]We do run such a preprocessing on the configuration.

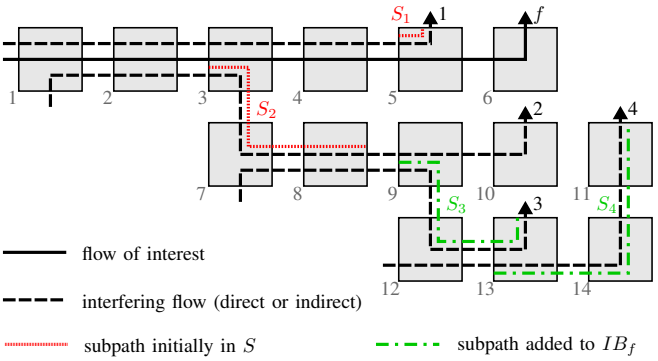

Fig. 6. Several contending flows and the subpaths with f as the flow of interest. Green : subpaths included in IB_f after the algorithm ; red : subpaths of DB_f.

A. Computing Direct Blocking Latency

The maximum direct blocking latency, part of the maximum service latency defined in Eq. (2), is defined in the following Theorem.

Theorem 3. *(Maximum Direct Blocking Latency) The maximum direct blocking latency for a f.o.i f along its path $\mathbb{P}_f$, in a NoC under flit-level preemptive FP multiplexing with strict service curve nodes of the rate-latency type $\beta_{R,T}$ and leaky bucket constrained arrival curves $\alpha_{\sigma,\rho}$ is equal to:*

$$T_{\mathbb{P}_f} + T_{hp} + T_{sp} + T_{lp}$$

with:

$$T_{\mathbb{P}_f} = \sum_{r \in \mathbb{P}_f} T^r \tag{5a}$$

$$T_{hp} = \sum_{i \in DB_f \cap hp(f)} \frac{\sigma_i^{cv(i,f)} + \rho_i \cdot \sum_{r \in \mathbb{P}_f \cap \mathbb{P}_i} \left(T^r + \frac{L^r_{slp(f)}}{R^r}\right)}{R_f} \tag{5b}$$

$$T_{sp} = \sum_{i \in DB_f \cap sp(f)} \frac{\sigma_i^{cv(i,f)} + \rho_i \cdot \sum_{r \in \mathbb{P}_f \cap \mathbb{P}_i} \left(T^r + \frac{L^r_{slp(f)}}{R^r}\right)}{R_f} \tag{5c}$$

$$T_{lp} = \sum_{r \in \mathbb{P}_f} \frac{L^r_{slp(f)}}{R^r} \tag{5d}$$

where:

$$L^r_{slp(f)} = \max\left(\max_{j \in sp(f)} \left(L_j \cdot \mathbf{1}_{\{sp(f) \supset r\}}\right), S_{flit} \cdot \mathbf{1}_{\{lp(f) \supset r\}}\right)$$

$$R_f = \min_{r \in \mathbb{P}_f} \left\{ R^r - \sum_{j \ni r, j \in shp(f)} \rho_j \right\}$$

Proof. The main idea is to integrate the impact of the flow serialization phenomena on the granted end-to-end service curve for the f.o.i f along its path $\mathbb{P}_f$. To achieve this aim, we adapt the results of the existing Theorem 2, based on the PMOO principle, to take into account the specificities of wormhole NoCs, in comparison to classic switched networks.

The wormhole NoCs allow the flit-level preemption during transmission, which modifies the lower priorities impact on the f.o.i in comparison to the non-preemptive mode in classic switched networks. Hence, a lower priority flow that is being transmitted at any node can delay the f.o.i f by at most the maximum transmission time of one flit. Consequently, the term $\max_{i \ni k, i \in slp(f)} L_i$ in Eq. (1b) must be modified for each node on $\mathbb{P}_f$ as follows:

- if there is one or more same-VC contending flow(s), this term becomes the maximum packet size of the contending, same priority flow(s) ;
- if there is one or more lower-VC flow(s), it equals the size of one flit S_{flit} ;
- if there is no same or lower-VC flow, it equals zero.

Therefore, the flit-level preemption property of NoCs infers that a f.o.i f will suffer from lower priority flows within any crossed node $r \in \mathbb{P}_f$ during the maximum transmission time of $L^r_{slp(f)}$, which is defined as follows:

$$L^r_{slp(f)} = \max\left(\max_{j \in sp(f)} \left(L_j \cdot \mathbf{1}_{\{sp(f) \supset r\}} \right) \ , \ S_{flit} \cdot \mathbf{1}_{\{lp(f) \supset r\}} \right)$$

Afterwards, we apply Theorem 2 while taking into account such modification (the impact of lower priority flows due to flit-level preemption). In doing this, we obtain the end-to-end service curve of flow f along its path $\mathbb{P}_f$, which integrates only the impact of the direct blocking set of f, DB_f, as follows:

$$R_f = \min_{r \in \mathbb{P}_f} \left\{ R^r - \sum_{j \ni r, j \in shp(f)} \rho_j \right\} \tag{7a}$$

$$T_f = \sum_{r \in \mathbb{P}_f} \left(T^r + \frac{L^r_{slp(f)}}{R^r} \right) + \sum_{i \in DB_f \cap shp(f)} \frac{\sigma_i^{\mathrm{cv}(i,f)} + \rho_i \cdot \sum_{r \in \mathbb{P}_f \cap \mathbb{P}_i} \left(T^r + \frac{L^r_{slp(f)}}{R^r} \right)}{R_f} \tag{7b}$$

Hence, as we can notice, the computation of the different parts of the direct blocking latency defined in Eq. (2) is straightforward.

□

To illustrate this computation, we take the example configuration of Figure 6. As we noticed earlier, all flows have the same priority and we furthermore assume they all have the same initial arrival curves $\alpha_{\sigma,\rho}$. Thus we have $T_{lp} = T_{hp} = 0$. We assume that for any node r, $T_r = T$ and $R^r = R$. We immediately have $T_{\mathbb{P}_f} = 6T$

The sum at the numerator of T_{sp} equals $2T$ for flow 1 (resp. $4T$ for flow 2) as it shares 2 (resp. 4) node outputs with f.

The rate $(\rho_i)_{i=1,2}$ is also constant. Thus R_f can be computed at node 1 or 2, where f suffers from contention from both flows 1 and 2, and we have $R_f = R - 2\rho$.

We now need the value of the burst for each flow of DB_f at the converging node with f. The converging node with f is node 1 for both flows 1 and 2, We get σ_1^1 and σ_2^1 either directly, if we assume flow 1 and 2 start at node 1, either by computing with the same method the service curve for these flows before node 1, and using the $\oslash$ deconvolution operator to derive their arrival curve at node 1.

B. Computing Indirect Blocking Latency

The maximum indirect blocking latency, part of the maximum service latency defined in Eq. (2), is defined in the following Theorem.

Theorem 4. *(Maximum Indirect Blocking Latency)*
The maximum indirect blocking latency for a f.o.i f along its path $\mathbb{P}_f$, in a NoC under flit-level preemptive FP multiplexing with strict service curve nodes of the rate-latency type $\beta_{R,T}$ and leaky bucket constrained arrival curves $\alpha_{\sigma,\rho}$, is as follows:

$$T_{IB} = \sum_{(k,subP) \in IB_f} \frac{\sigma_k^{subP[0]}}{\widetilde{R}_k^{subP}} + \widetilde{T}_k^{subP} \tag{8}$$

where:

$$\widetilde{R}_k^{subP} = \min_{r \in subP} \left\{ R^r - \sum_{j \ni r, j \in hp(f)} \rho_j \right\} \tag{9a}$$

$$\widetilde{T}_k^{subP} = \sum_{r \in subP} \left(T^r + \frac{S_{flit} \mathbf{1}_{\{lp(k) \supset r\}}}{R^r} \right) + \sum_{i \in DB_k^{|subP} \cap hp(k)} \frac{\sigma_i^{cv(i,k)} + \rho_i \sum_{r \in subP \cap \mathbb{P}_i} \left(T^r + \frac{S_{flit} \mathbf{1}_{\{lp(k) \supset r\}}}{R^r} \right)}{\widetilde{R}_k^{subP}} \tag{9b}$$

Proof. The maximum indirect blocking latency for a f.o.i f induced by the flows set IB_f, computed in the previous section using Algorithm 1.

Any flow $j \in IB_f$ will impact the f.o.i f during the maximum time it occupies the associated subpath $subP_j$, Δt_j^{max}. Hence, a safe upper bound on the indirect blocking latency is as follows:

$$T_{IB} \leq \sum_{j \in IB_f} \Delta t_j^{max}$$

On the other hand, for any flow $j \in IB_f$, Δt_j^{max} is upper bounded by the end-to-end delay bound of flow j along its associated subpath $subP_j$, $D_j^{subP_j}$, which infers the following:

$$T_{IB} \leq \sum_{(j,subP_j) \in IB_f} D_j^{subP_j} \tag{10}$$

Based on Theorem 1, the delay bound of flow j, $D_j^{subP_j}$, is computed as the maximum horizontal distance between:

- the maximum arrival curve of flow j at the input of the subpath $subP_j$, $\alpha_j^{subP_j[0]}$, which takes into account the impact of all the interferences suffered by flow j upstream the node $subP_j[0]$, i.e., the propagated arrival curve of flow j until the input of $subP_j[0]$ using Theorem 1;
- the granted service curve to flow j by its VC along $subP_j$, when ignoring the same-priority flows (which are

already included in IB_f), $\widetilde{\beta}_j^{subP_j}$. The latter condition is due to the pipelined behavior of the network, where the same-priority flows sharing $subP_j$ are served one after another if they need shared resources. Hence, the impact of the same-priority flows than flow j is already integrated within the sum expressed in Eq. (10).

To compute the granted service curve $\widetilde{\beta}_j^{subP_j}$ for each flow $j \in IB_f$ along $subP_j$, we follow similar approach than in the proof of Theorem 4 through applying the existing Theorem 2, when:

- ignoring the same-priority flows in $sp(j)$, thus all $shp(j)$ will become $hp(j)$ and $slp(j)$ will become $lp(j)$ in Eqs. (1 a) and (1 b);
- considering the flit-level preemption, thus the impact of lower-priority flows in Eq. (1 a) is bounded by the maximum transmission time of $S_{flit} \cdot \mathbf{1}_{\{lp(k) \supset r\}}$ within each crossed node $r \in subP_j$;
- considering only the direct blocking flows of j intersecting $\mathbb{P}_j$ on $subP_j$, thus considering $DB_j^{|subP_j} \cap hp(j)$ in Eq. (1 b).

Hence, we obtain $\widetilde{R}_j^{subP_j}$ and $\widetilde{T}_j^{subP_j}$ described in Eqs. (9a) and (9b), respectively. Consequently, the maximum indirect blocking latency in Eq. (10) can be re-written as follows:

$$T_{IB} \leq \sum_{(j,subP_j) \in IB_f} \frac{\sigma_j^{subP_j[0]}}{\widetilde{R}_j^{subP_j}} + \widetilde{T}_j^{subP_j} \qquad (11)$$

□

Here again, we propose an application to the configuration on Figure 6. Since all flows have the same priority, for all $j, subP_j \in IB_f$, $R_j^{subP_j}$ and $T_j^{subP_j}$ are immediately obtained. What is left is to compute the burst of j at the first node of the subpath $subPj$. Here, it is pretty straightforward because the corresponding flows start at this node and we have $\sigma_j^{subP_j[0]} = \sigma$. On a more complex situation, we can perform for each j the same analysis on the section before $subP_j[0]$, but this time ignoring the flow of interest f in all the analysis. This is due to the fact that we are doing this computation to derive f worst-case latency bound. Therefore we must not consider that f can block a flow that will in turn block f.

VI. Performance Evaluation

In this section, we first analyse the derived delay bounds using our proposal and conduct a comparative analysis with the existing approaches, based on Scheduling Theory [3] [2] as well as CPA [6]. Then, we perform experiments on a manycore platform, to assess our delay bounds tightness with reference to the experimental results.

A. Comparative Study

To compare our proposal to the most relevant approaches relatively to our context (namely [2], [6]), we consider the example used in [2], which is illustrated in Figure 7.

The parameters for the flows are gathered in Table III. Using our approach, the end-to-end delay bound of flow 3 is 44

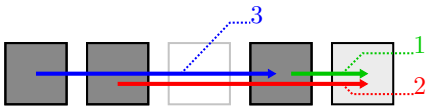

Fig. 7. A simple configuration from [2].

Flow index	1	2	**3**
Priority	1	2	**3**
Period	100	100	**100**
Deadline	100	100	**40**
Release jitter	0	0	**0**
Base latency (no contention)	21	24	**14**
Packet size	19	20	**10**
Cycle accurate scenario in [2]	21	43	**43**
Upper bound by [3]	21	45	**38**
Upper bound (our approach)	23	57	**44**

TABLE III
Minimal example

cycles. However, the delay bound predicted by the model in [3] is only 38, and the computed bound in [2] is 43. This result shows that our proposed timing analysis guarantees safe bounds on a configuration known to be problematic with some Scheduling Theory approaches. We would like to conduct further comparaison with such approaches using large-scale configuration, like the one used in [5], but unfortunately it is almost impossible to reproduce the analyzed scenarios due to the lack of information on the considered platform and flows characteristics in [5].

The second performed comparison is based on the configuration presented on Figure 8 taken from [6] using CPA approach, where all flows have a packet length 4 flits. We have reproduced different scenarios of [6] to compute the delay bounds with our proposal with respect to the flow rate and buffer size.

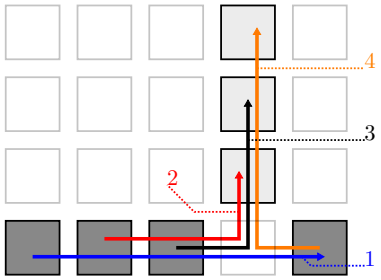

Fig. 8. A simple configuration from [6].

First, we vary the requested bandwidth per sender (*i.e.* the rate of each flow relatively to the maximal rate). Since the packet length is constant, we adjust the flow period to get different values of rate. The full bandwidth corresponds to a rate of one flit per cycle.

For each bandwidth value, we compute the corresponding delay bound predicted by our model for all 4 flows for a buffer size equal to 4 flits and the derived results are in Figure 9. For each flow, we also plotted a vertical line representing the saturation point of [6] CPA model : when the CPA-predicted latency is greater than 10^3 cycles[7], we consider that the model

[7]This value is similar in this case to infinity since it is very high in comparison to the flows deadline

diverges.

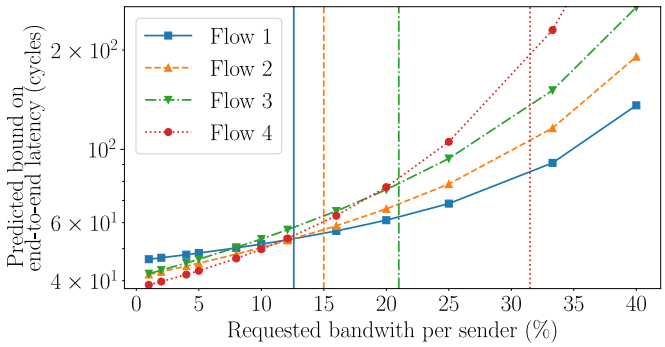

Fig. 9. Predicted bounds for different values of bandwidth

We first notice that the curves of our predictions are smoother than [6]. Moreover, for low bandwidths (below 10%), our predictions are similar to [6], or even tighter. They also grow smoother for higher bandwidths and do not present any saturation point like in [6]. Specifically, for flow 1, which may suffer from buffer backpressure, the CPA approach predicted upper bound reaches 10^3 cycles shortly after 12.5% bandwidth. Our bound, on the other hand, is 54 cycles for 12.1% and 57 cycles for 16% bandwidth, very tight in comparison to the simulation results in [6].

Next, we study the impact of buffer size with a constant requested bandwidth per sender (12.5%) for flow 1. We compute the predictions of our model for the same buffer sizes in the experiment by [6] and for additional values, especially for all buffer sizes lower than packet size which are not handled in the CPA model [6]. The derived results of both approaches are illustrated in Figure 10.

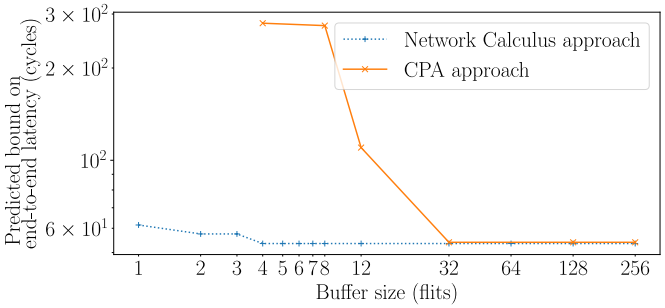

Fig. 10. Delay bounds of flow 1 vs buffer size under CPA and NC approaches

As we can notice, our approach allows much tighter delay bounds for small buffer sizes. For buffer sizes lower than 12 flits (3 packets), we obtain a bound tightness improvement of 45.5% with our model in comparison to the CPA one, and it is more than 80% for the lowest buffer size (4 flits). We also notice that increasing buffer size does not improve the delay bound past a certain point under our approach. For instance, the end-to-end delay upper bound remains constant for buffer sizes above 4 flit (the size of one packet for any flow in the configuration). This is due to the fact that the spread index of a flow remains constant when the buffer size exceeds the length of one packet. With a constant spread index, the indirect blocking set analysis remains the same; thus the indirect blocking latency (and the end-to-end delay bound).

B. Experiments on a Physical Platform

1) Setting Data Flow Configurations: We conducted experiments on a TILE-Gx8036, a 36-core chip developed by Tilera.[8] Our aim was to determine whether our model could actually be used to predict worst-case end-to-end latencies on a physical manycore platform. The platform features a NoC (also called "UDN", User Dynamic Network) with no virtual channels. It uses X-Y routing and the buffers can contain 3 flits. We have tested several configurations. Each flow transmitted at least 20000 packets[9] and we sampled 9999 of them after a warm-up time.

To create flows with arbitrary parameters, we wrote a code running on the Tilera platform. The code assigns cores to a specific task (sending or receiving packets), depending on the parameters gathered in a configuration file that is loaded onto the TILE-Gx before running the application. Each packet producer sends arbitrary packets of fixed size with a minimal inter-packet time.

2) Latency Measurements: To measure packet end-to-end latency for each flow, we proceed as follows. Each TX (resp. RX) process samples the cycle counter right before sending a packet on the UDN (resp. right after receiving a packet from the UDN) and stores the value in an array, at the position corresponding to the packet number. We print and process these values after the execution to get the measured end-to-end latency for each packet. We consider the 99.99%-accurate measured bound, *i.e.* the latency such that 99.99% of the flows have a latency below this value.

However, the measured latency includes the time needed by the application to access the UDN, at TX and RX ends. This latency is not taken into account in our model. To determine it, we measure end-to-end latency on a 15-flow configuration where the UDN latency is known (no congestion). We ran the experiment 100 times, and we found that the minimal UDN access latency is a piecewise affine function of the packet length, and that 99.99% of the packets are at most 5 cycles above the minimal UDN access latency.

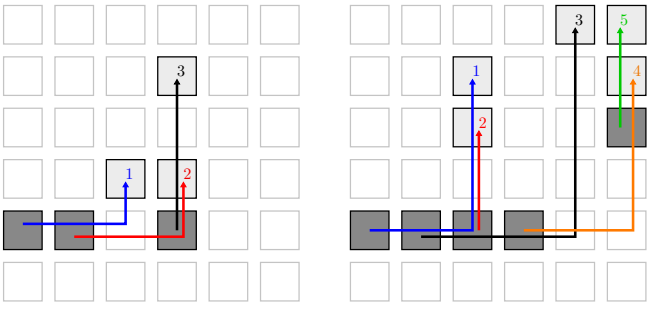

Fig. 11. Experiment configurations 1 (left) and 2 (right)

3) Experiments and Discussion: We test two configurations, shown in Figure 11. The first configuration is a simple

[8]http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx36.pdf

[9]The lowest-rate flow transmits 20000 packets, others transmit packets as long as the lowest-rate flow is transmitting.

one with only 3 flows, with a period of 200 cycles and packets of 8 flits. In this configuration, flow 1 can undergo indirect blocking from flow 3. The second one has 5 flows. Flow 1 can experience indirect blocking from flow 4 via flow 3. Flow 3 can experience indirect blocking from flow 2 via flow 1. All flows are 100-cycle-periodic and their packets are 8-flit-long.

For each flow, we compute the theoretical bound on end-to-end latency and the 99.99%-accurate measured bound. The derived results are in Fig. 12.

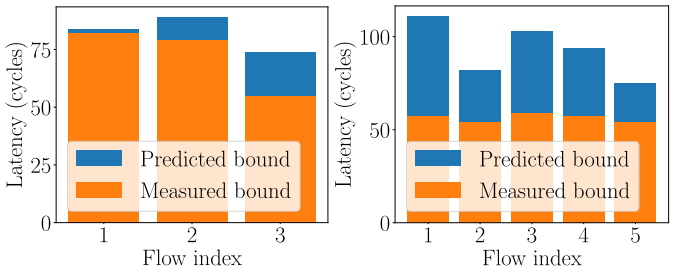

Fig. 12. Experiment results for configuration 1 (left) and 2 (right)

We notice that for the simple configuration of 3 flows, our delay bounds are tight, in comparison to the measured ones. For instance, the tightness bound for flow 1 is 97.6%. However, for the more complex configuration (5 flows), our analytical delay bounds are less tight, especially for flows 1 and 3, that are subject to more complex indirect blocking. This fact is mainly due to the difficulty of catching the theoretical worst-case scenario, which requires all interfering flows to be synchronized in an unfavorable way along the shared paths. Although this may seem counter-intuitive, the more flows are involved in one congestion pattern, the more difficult it will be to reach a proper synchronization between the interfering flows at least once during the experiments.

VII. Conclusion and Further Work

We have proposed an approach based on Network Calculus to address main limitations of recent works in real-time NoC analysis. Specifically, our model applies to a wide range of architectures : it supports multiple VCs, priority sharing and VC-sharing. Our modeling choices are also fine-grained. We take into account buffer size influence by studying the way packets can spread in the network, thus providing fine, safe bounds on backpressure-due delays. We use recent contributions in Network Calculus Theory to refine the analysis by accounting for serialization effects, which to the best of our knowledge, has not been done on NoCs yet. The result is a generic, scalable model that provides safe bounds and noticeable improvements, in comparison to existing approaches, e.g. Scheduling Theory and CPA, and with reference to experimental results.

There are mainly three axes that we would like to explore. First, we would like to perform a more thorough sensitivity analysis of our model. Specifically, we would like to study different types of traffic to get a grasp on the influence of flow burstiness and rate, maximum network utilization rate and buffer size on the predicted latency. There is also room for refining the model even more, *e.g.* specifying arbitration policies between classes within the same virtual channel and between flows of the same class. We could also evaluate the likelihood of indirect blocking scenarios to get a better interpretation of experimental results. Finally, we want to develop further the testing process on a manycore platform. We need to design a better test bench that is less sensitive to non-NoC-related influences during the application execution, such as memory access latencies or interruptions.

Acknowledgements

We would like to thank Stephen Mallon (University of Sydney) for his fantastic expertise on the Tilera platform we used, as well as Guillaume Jourjon (Data61 – CSIRO, Sydney) and Vincent Gramoli (University of Sydney) for their friendly supervision and help on the experiments. Our gratitude also goes to Emmanuel Lochin (ISAE – Université de Toulouse) for setting up the collaboration with Data61 and University of Sydney.

References

[1] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, pp. 62–76, Feb 1993.

[2] Q. Xiong, F. Wu, Z. Lu, and C. Xie, "Extending real-time analysis for wormhole nocs," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2017.

[3] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pp. 161–170, April 2008.

[4] Z. Shi and A. Burns, "Real-time communication analysis with a priority share policy in on-chip networks," in *2009 21st Euromicro Conference on Real-Time Systems*, pp. 3–12, July 2009.

[5] M. Liu, M. Becker, M. Behnam, and T. Nolte, "Tighter time analysis for real-time traffic in on-chip networks with shared priorities," in *10th IEEE/ACM International Symposium on Networks-on-Chip*, August 2016.

[6] S. Tobuschat and R. Ernst, "Real-time communication analysis for networks-on-chip with backpressure," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 590–595, March 2017.

[7] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, (San Jose, CA, USA), EDA Consortium, 2015.

[8] Y. Qian, Z. Lu, and W. Dou, "Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pp. 44–53, May 2009.

[9] F. Jafari, Z. Lu, and A. Jantsch, "Least upper delay bound for vbr flows in networks-on-chip with virtual channels," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, pp. 35:1–35:33, June 2015.

[10] A. Mifdaoui and H. Ayed, "Buffer-aware worst case timing analysis of wormhole network on chip," *arXiv*, vol. abs/1602.01732, 2016.

[11] B. Nikolić, H. I. Ali, S. M. Petters, and L. M. Pinho, "Are virtual channels the bottleneck of priority-aware wormhole-switched noc-based many-cores?," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, 2013.

[12] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Heidelberg: Springer-Verlag, 2001.

[13] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic, "Improving performance bounds in feed-forward networks by paying multiplexing only once," in *14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, pp. 1–15, March 2008.