Work-in-Progress: Extending Buffer-Aware Worst-Case Timing Analysis of Wormhole NoCs

Frédéric Giroudot, Ahlem Mifdaoui

Work-in-Progress: Extending Buffer-Aware Worst-Case Timing Analysis of Wormhole NoCs

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*—Worst-case timing analysis of Networks-on-Chip (NoCs) is a crucial aspect to design safe real-time systems based on manycore architectures. In this paper, we present some potential extensions of our previously-published buffer-aware worst-case timing analysis approach to cope with bursty traffic such as real-time audio and video streams. A first promising lead is to improve the algorithm analyzing backpressure patterns to capture consecutive-packet queueing effect while keeping the information about the dependencies between flows. Furthermore, the improved algorithm may also decrease the inherent complexity of computing the indirect blocking latency due to backpressure.**

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

I. Context and Related Work

Wormhole Networks-on-chip (NoC) have become a standard interconnect for manycore architectures because of their high throughput and low latency capabilities. Packets transmission is done by splitting them into constant length words called flits, then forwarding each flit from router to router, without having to wait for the remaining flits at each hop. The counterpart is that contention in the network can induce complex backpressure[1] patterns that are difficult to predict. Hence an appropriate timing analysis of NoCs is crucial for real-time applications.

Proposed approaches mainly include Scheduling Theory-based models [1]–[3], CPA-based models [4], [5] and Network Calculus-based models [6]–[9]. Scheduling Theory (ST) was used in particular in [1], which was found to be optimistic in specific cases and further corrected in [2]. This last work, however, does not consider priority-sharing. A recent work [3] combines ST with Recursive Calculus, but focuses only on Round-Robin arbitration.

Compositional Performance Analysis (CPA) was used in [4] to develop a model supporting priority-sharing and Virtual Channel (VC)-sharing but ignoring the buffer backpressure. The latter has been addressed in [5]. However the proposed analysis is limited to NoCs with a single VC and buffers larger than the size of one packet.

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

Other works using Network Calculus (NC) [10] include [7] and [8], but they consider that backpressure does not occur in the NoC.

In a previous work [9] (referred to as RTAS18 hereafter), we aimed at overcoming the aforementioned limitations and presented a model based on NC allowing to (i) analyse common NoC architectures, supporting priority-sharing, multiple VCs, arbitrary service policies; (ii) account for limited buffer size in NoCs and the impact of the resulting backpressure on end-to-end latencies; (iii) reduce pessimism of delay bounds by taking into account flow serialization.

Most of the existing approaches taking into account backpressure consider only Constant Bit Rate (CBR) traffic, *i.e.* one fixed-length packet within a minimum inter-arrival time. However, there are some traffic types, such as real-time audio, video and bursty data streams, which do not fulfill the CBR model.

Contributions: In this work, we present some potential extensions of our worst-case timing analysis RTAS18 to cope with bursty traffic.

The remainder of this paper is organized as follows. In Section II, we recall and detail our previous approach (II-A,II-B) and illustrate the method on an example (II-C). We then propose leads to extend the approach in Section III, starting from an example to motivate further enhancements (III-A) and exploring two aspects to work on (III-B and III-C). Section IV concludes the paper.

II. Recap of Buffer-Aware Approach

A. System Model

We consider the NoC as a set of interconnected input-buffered routers with Virtual Channels (Figure 1). Each output of a router is connected to one input of another router. In the case of 2D-mesh NoCs, routers have 5 inputs and 5 outputs, referred to as North, South, West, East and Local.

We use NC to model routers and traffic. We model each router-output pair r, referred to as a *node*, as a multiplexer, with a total processing ability represented by its minimal service curve:

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

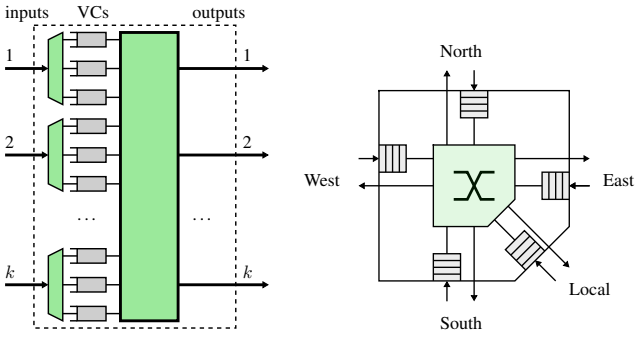

Fig. 1. Router architecture.

R^r represents the processing capacity of the router in flits per cycle, T^r corresponds to the processing delay (the delay a flit experiences when it is processed). Similarly, we model each flow with its *maximal arrival curve*:

$$\alpha_f = \sigma_f + \rho_f t$$

It depends only on the flow characteristics: periodicity (or minimal inter-arrival time), packet length and overhead length, release jitter (if any). We also denote its path (the list of the nodes it crosses from source to destination) as $\mathbb{P}_f$.

B. Main Steps of Analysis

To compute the end-to-end latency bound for a flow of interest (*foi*) f, we first compute its initial arrival curve α_f. Then, we compute its *end-to-end service curve* along its path. It is denoted β_f and depends on the contention induced by other flows that the *foi* can undergo. Finally, knowing these two curves, we use one of the fundamental results in NC to derive a bound on the worst-case end-to-end latency, which is the maximal horizontal distance between the arrival and service curves.

The computation of the end-to-end service curve consists of 4 steps:

1) We compute the "base latency", *i.e.* the technological latency to forward a packet when there is no congestion;
2) We compute the maximum delay induced by flows that directlty block f, *i.e.* flows that share resources with f along their paths. This delay is called "direct blocking delay" and denoted T_{DB};
3) We determine the *indirect blocking set* (IB set) of f, denoted IB_f. This set contains {flow index, subpath} pairs corresponding to flows that may induce backpressure on the *foi* when the specified subpath holds a packet or part of a packet. This is the step where we account for the limited buffer size. We will detail it in Section II-C;
4) We use IB_f to derive a bound on the maximum delay induced by flows that have at least one {flow index, subpath} pair in IB_f. This delay is called "indirect blocking delay" and denoted T_{IB}.

Only steps 3 and 4 rely on the CBR traffic assumption and will be impacted by our model extension. Thus we will not develop steps 1 and 2 any further. We will illustrate the computation of IB set through an example in the next section. More details can be found in [9].

C. An Illustrative Example

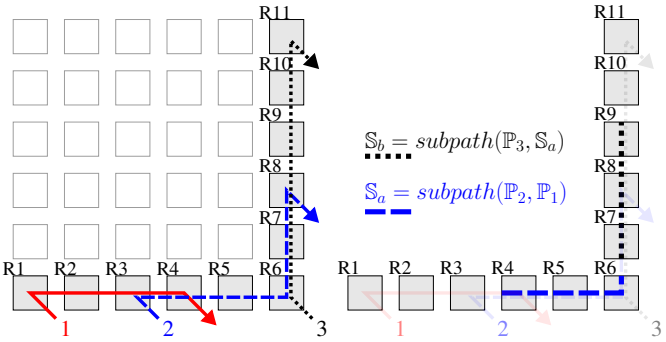

Fig. 2. Example configuration (left) and subpath computation (right)

To take into account the buffer size when performing the IB set computation, we focus on the point where two flows sharing part of their paths diverge (divergence point), *i.e.* the point from where those flows do not compete for the same outputs any more. We look at how much packets can spread in the network after the divergence point and still impact the flow of interest. This critical section, where the head of a packet is located *after* a divergence point while part of that packet still uses shared resources, is called a *subpath*. It depends on the packet length and buffer size. We denote such a subpath $\mathbb{S} = subpath(\mathbb{P}_k, \mathbb{S}_l)$ and call it "subpath of $\mathbb{P}_k$ relatively to $\mathbb{S}_l$". $\mathbb{S}$ is the subpath of flow k after the divergence point between $\mathbb{P}_k$ and $\mathbb{S}_l$ (that can be either the whole path or a subpath of flow l).

Consider the 3-flow configuration on a 6×6 mesh NoC, represented on Figure 2, to illustrate the IB set computation. We assume each buffer can hold one flit and all flows have 3-flit packets, so that each packet can be stored in three buffers. We also suppose there is only one priority level and a single VC. Furthermore, we consider all nodes have the same service curve and all flows have the same initial arrival curve.

We first compute subpaths of contending flows relatively to the *foi*, flow 1. As only flow 2 shares resources with flow 1, there is only one such subpath, which we denote $\mathbb{S}_a$. It is 3-node long and can contain at most a 3-flit packet of flow 2. Then, we iterate on the newly found subpath(s). We find that the path of flow 3 shares resources with $\mathbb{S}_a$, so we compute the corresponding subpath of flow 3 relatively to $\mathbb{S}_a$, that we denote $\mathbb{S}_b$.

We only keep subpaths of flows that do not share resources with the *foi*. Thus we finally have:

$$IB_1 = \{\{3, \mathbb{S}_b\}\}$$

We can then compute T_{IB} using Theorem 4 in [9]. This step may induce recursive calls to compute needed intermediate arrival curves. We will detail this in Section III-C.

III. Potential Extensions of Buffer-Aware Approach

A. Motivations

Consider the following example (Figure 3). It is similar to the previous one, but the convergence point between flow 2

and 3 is changed: flow 3 starts at node R7 instead of R6.

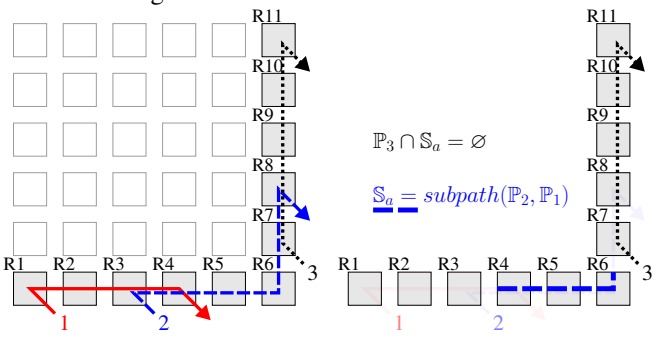

Fig. 3. A second example (left) and its subpath computation (right)

When we compute the IB set of the *foi* (flow 1), IB_1, we find that the computed subpath of flow 2, $\mathbb{S}_a$, does not intersect any other flow path. This is due to the fact that in this configuration, the convergence point of flows 2 and 3 is one hop further than previously, while the way packets may spread in the network does not change. Hence, if a packet of flow 2 is blocked by a packet of flow 3, then the packet of flow 2 must have its head flit right before the convergence point of flows 2 and 3. This implies the tail flit of flow 2 will be downstream of the divergence point of flows 1 and 2. Thus, it cannot block flow 1 and flow 3 is not in IB_1:

$$IB_1 = \{\}$$

However, this result relies on the assumption that there is only one packet of flow 2 in the network. Consider the scenario depicted in Figure 4. We suppose a packet A of flow 3 is being transmitted. It requested the North output port of R7 and was granted access. Now its header flit has reached the input port of R8. At the same time, there is a packet B of flow 2 which header flit has reached node R7 input. It also requested the North output of R7, but as A is using it, it has to wait. B is immediately followed by another packet of flow 2, denoted C, which header flit is in R4. C previously requested R3 East output port and was granted it. Here, C has not been completely injected into the network. In fact, B has its last flit in R5 buffer, and is blocked. Consequently, C has to wait that B moves forward to be able to move too. Finally, flow 1 has injected a packet D into the NoC. D has its header flit in R3 and is requesting R3 East output port. However, as C already requested and got the use of that same port, D has to wait.

In that case, D is indireclty blocked by A. This means that under bursty traffic assumption, flow 3 may impact flow 1, which was not detected when computing IB_1.

B. Improving Indirect Blocking Set Computation

The RTAS18 approach takes into account the way *one* packet spreads in the NoC. Consequently, as seen on the previous example, we may not include all flows inducing backpressure in the IB set under bursty traffic assumption.

An idea to cope with this assumption is to consider several consecutive subpaths of one flow. We do so by allowing the algorithm to compute the subpath of a flow relatively to a

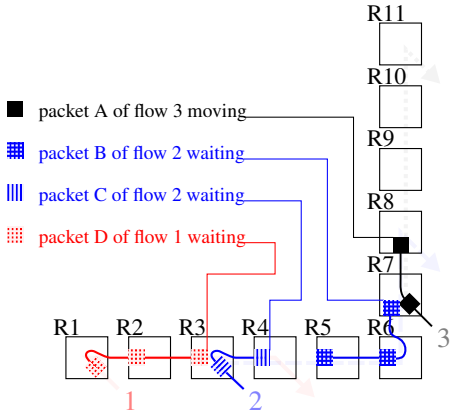

Fig. 4. Packet configuration with two instances of flow 2

subpath of the same flow. An illustration of this extension on the example of Figure 3 is given on Figure 5. We would add a subpath of flow 2 right after the existing one ($\mathbb{S}_a$) to model the possibility that there may be several packets of flow 2 queueing in line. This subpath would be $\mathbb{S}_b$, and it intersects the path of flow 3. Then we would necessarily compute subpath $\mathbb{S}_c$ as the subpath of flow 3 relatively to subpath $\mathbb{S}_b$. This way, we can model scenarios such as the one presented in Figure 4, where traffic is assumed to be bursty and consequenlty there are consecutive packets of one flow queueing in the NoC.

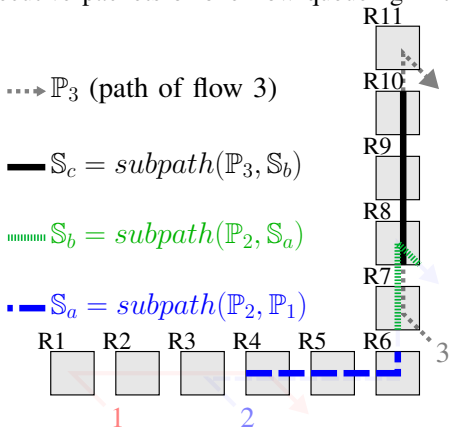

Fig. 5. Taking into account consecutive packets by computing several consecutive supaths of one flow

If we take into account the additional subpath of flow 2, the IB set of *foi* 1 becomes:

$$IB_1 = \{\{3, \mathbb{S}_c\}\}$$

And we can account for the extra indirect blocking delay undergone by flow 1 because of flow 3.

The main drawback of this extension is that, if we have several flows with overlapping paths, we may be forced to compute a lot of overlapping subpaths and end up with an IB set that might depend on the order in which we compute the subpaths relatively to other subpaths, or that contains duplicates. Indeed, using the algorithm we proposed in [9] (even with the extension) does not allow to keep track of the dependencies between subpaths.

This fact will likely increase the number of computed subpaths, thus it could increase pessimism of computed indirect blocking delay. To handle such an issue, we will improve the subpath representation and maintain information about their dependencies using formal representations such as trees or another dependency-aware structure. This should allow us to model indirect congestion patterns in a more accurate way and avoid the over-pessimism of the computed T_{IB}. Computation of the IB set may suffer from an additional complexity, but it is hard to quantify without implementing the method.

C. Refining Indirect Blocking Delay

With the RTAS18 approach, the indirect blocking delay T_{IB} of the *foi* depends on the intermediate arrival curves at the input of each subpath in its IB set. These intermediate curves account for the impact of the delay upstream the first node of each subpath in the IB set on the burstiness of the traffic. At the same time, it induces recursive computations of intermediate service curves; thus a high computational complexity.

To show this, we get back to the first example in Figure 2 and we compute T_{IB} with RTAS18 approach using Theorem 4 of [9]. The indirect blocking delay induced by flow 3 on flow 1 is bounded by the horizontal distance between:

- the maximum arrival curve of flow 3 at the input of the subpath $\mathbb{S}_b$ – this arrival curve takes into account the impact of all the interferences suffered by flow 3 upstream the node R7, *i.e.*, it is the propagated arrival curve of flow 3 until the input of R7;
- the granted service curve to flow 3 by its VC along $\mathbb{S}_b$, called VC-service curve, when ignoring the same-priority flows. The latter condition is due to the pipelined behavior of the network: if there are same-priority flows sharing $\mathbb{S}_b$ resources with flow 3 (here, there are none of them), they are served one after another. Hence, their impact is already integrated.

To compute the arrival curve of flow 3 at the input of R7, we need to compute the service curve granted to flow 3 on the section [R6]. Furthermore, to compute such a service curve, we need to know the arrival curve of flow 2 at the input of R6. Thus, we need to compute the service curve of flow 2 granted from its source until the input of R6, *i.e.* [R3,R4,R5]. For this, we need to know the arrival curve of flow 1 at the input of R3; thus we need the service curve of flow 1 on [R1, R2]. Hence, there is a recursive call to the function computing service curves.

However, when considering the proposed extension allowing multiple subpaths per flow (as many as there can be simultaneous packet instances of this flow), we would already take the propagated burstiness into account. Thus we would not need to get the equivalent burstiness of the traffic with the intermediate arrival curve. Instead, we may use, at the beginning of each subpath, the initial arrival curve of the corresponding flow for one packet instance. Determining such an initial arrival curve is an explicit computation with a constant complexity, whereas computing an intermediate arrival curve may induce recursive calls.

Hence, a major expected benefit of this extension would be to decrease computational complexity of the indirect blocking delay computation step, since we would not need recursive calls any more.

In the aforementioned example of Section II-C, with the RTAS18 approach, three recursive calls are needed to get the arrival curve of flow 3 at the beginning of subpath $\mathbb{S}_b$. These three calls would be avoided when considering the proposed extension.

This aspect has to be further investigated to accurately assess its impact on the overall complexity of the buffer-aware approach. It could also counter-balance the additional algorithmic complexity of the subpath computation step.

IV. Conclusion

In this paper, we have presented an extension opportunity for a previous published work addressing worst-case timing analysis of wormhole Networks-on-Chip to support bursty traffic. After examining the impact of such an assumption on our model, we came up with two promising leads. First, we will extend the indirect blocking set computation algorithm to capture consecutive-packet queueing effect, while keeping the information about the dependencies between subpaths. Second, the inherent complexity to compute the indirect blocking latency will be decreased by avoiding recursive computation of arrival and service curves. In a near future, we plan to explore more thoroughly these two leads and formally analyze the underlying concepts, as well as evaluate the gains or losses in terms of bound tightness and computational complexity.

References

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

[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] M. Liu, M. Becker, M. Behnam, and T. Nolte, "Buffer-aware analysis for worst-case traversal time of real-time traffic over rra-based nocs," in *27th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, March 2017.

[4] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *Proceedings of Design, Automation Test in Europe Conference Exhibition*, 2015.

[5] S. Tobuschat and R. Ernst, "Real-time communication analysis for networks-on-chip with backpressure," in *Design, Automation Test in Europe Conference Exhibition*, 2017.

[6] 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, 3rd ACM/IEEE International Symposium on*, May 2009.

[7] 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.

[8] M. Boyer, B. Dupont De Dinechin, A. Graillat, and L. Havet, "Computing Routes and Delay Bounds for the Network-on-Chip of the Kalray MPPA2 Processor," in *ERTS 2018 - 9th European Congress on Embedded Real Time Software and Systems*, (Toulouse, France), Jan. 2018.

[9] F. Giroudot and A. Mifdaoui, "Buffer-aware worst-case timing analysis of wormhole nocs using network calculus," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, (Porto, PT), pp. 1–12, 2018.

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