On Routing Flexibility of Wormhole-Switched Priority-Preemptive NoCs

Borislav Nikolic

Luis Miguel Pinho

Leandro Soares Indrusiak

CISTER-TR-160602

On Routing Flexibility of Wormhole-Switched Priority-Preemptive NoCs

Borislav Nikolic, Luis Miguel Pinho, Leandro Soares Indrusiak

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Flit-level preemptions via virtual channels have been proposed as one viable method to implement priority preemptive arbitration policies in NoC routers, and integrate NoCs in the hard real-time domain. In recent years, researchers have explored several aspects of priority-preemptive NoCs, such as different arbitration techniques, different priority assignmen tmethods (where applicable) and different workload mapping approaches, all with the common objective to use interconnect mediums more efficiently. Yet, the impact of different routing techniques on such a model is still an unexplored topic. Motivated by this reality, in this work we study the effects of routing flexibility on wormhole-switched priority-preemptive NoCs.

On Routing Flexibility of Wormhole-Switched Priority-Preemptive NoCs

Borislav Nikolić
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto
Portugal
Email: borni@isep.ipp.pt

Luís Miguel Pinho
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto
Portugal
Email: lmp@isep.ipp.pt

Leandro Soares Indrusiak
Real-time Systems Group
Department of Computer Science
University of York
Email: lsi@cs.york.ac.uk

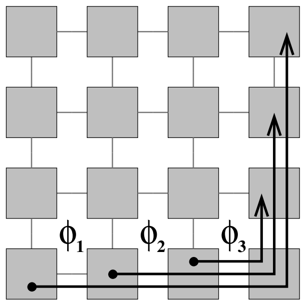
***Abstract*—Flit-level preemptions via virtual channels have been proposed as one viable method to implement priority-preemptive arbitration policies in NoC routers, and integrate NoCs in the hard real-time domain. In recent years, researchers have explored several aspects of priority-preemptive NoCs, such as different arbitration techniques, different priority assignment methods (where applicable) and different workload mapping approaches, all with the common objective to use interconnect mediums more efficiently. Yet, the impact of different routing techniques on such a model is still an unexplored topic. Motivated by this reality, in this work we study the effects of routing flexibility on wormhole-switched priority-preemptive NoCs.**

I. Introduction

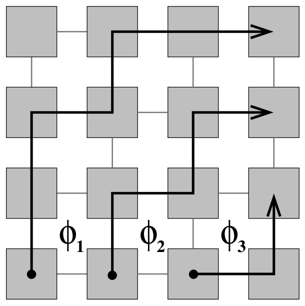
Slowly but steadily, many-core platforms pave their path into the real-time embedded domain. Traditionally, in the real-time analysis of many-cores, the emphasis is on one particular type of shared resources – processing elements. Yet, as the number of processing elements integrated within many-cores continues to grow every day, the contentions for other shared resources, such as the interconnect medium, become more apparent. Thus, in order to perform the end-to-end worst-case analysis of applications deployed upon a many-core platform, it is no longer sufficient to take into account only computation requirements, but delays of the communication and memory traffic have to be considered as well. This implies that the real-time analysis of interconnects is becoming an essential part of the real-time analysis of many-cores.

The Network-on-Chip architecture [1] (NoC) is the most prevalent choice for the interconnect medium in nowadays available many-cores (e.g. [2]–[4]), due to it's scalability potential [5]. Typically, the data transfer over NoCs is performed with *the wormhole switching* technique [6]. Some wormhole-switched NoCs (e.g. [2]) have *virtual channels* [7], [8], which means that data of multiple traffic flows can be simultaneously buffered in a single router port. This feature allows to perform *preemptions* among traffic flows [9] and implement priority-preemptive arbitration policies [10], which is a promising step towards deriving hard real-time guarantees for NoCs, and subsequently integrating them in the hard real-time domain (the main topic of interest in this work).

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by the European Union under the Seventh Framework Programme (FP7/2007-2013), grant agreement n° 611016 (P-SOCRATES).



(a) With X-Y routing



(b) Without X-Y routing

Fig. 1: Motivational example of 3 traffic flows

The majority of studies on wormhole-switched priority-preemptive NoCs have a common assumption that the deterministic dimension-ordered *X-Y routing policy* is applied. This technique is very popular among hardware vendors (e.g. [2], [4]), due to a relatively easy implementation and a deadlock-free property [11]. Yet, the X-Y routing technique has some disadvantages as well. For example, it has been demonstrated that this mechanism is not very efficient for memory responses on platforms where memory controllers are accessed from the topmost and the bottommost rows of tiles [12], [13]. Another disadvantage is demonstrated with an example from Figure 1. Consider a 2-D mesh NoC and three traffic flows ϕ_1, ϕ_2 and ϕ_3, with fixed sources and destinations. Let the no-load latency and the deadline of each traffic flow be 3 and 5 time units, respectively. Assuming the X-Y routing policy (Figure 1a), due to contentions, some flows will miss deadlines. Since meeting all deadlines is often an absolute imperative in the hard real-time domain, then it might be necessary to opt for a platform with sufficiently better hardware characteristics (e.g. higher router frequencies and/or link bandwidths), such that flow traversal times would be reduced to the extent that all deadlines are met. This solution could be very expensive.

Now, consider Figure 1b. Notice that each flow still travels across the same number of routers, thus the traversal latencies remain as in the previous case. But unlike before, in this case there are no contentions, and hence no missed deadlines. This implies that if we could (i) thoughtfully derive flow-paths, such that the inter-flow interference is minimised, and (ii) guarantee the conservation of the deadlock-free property, we could exploit the available NoC resources more efficiently. The benefits are twofold. First, significant design-cost reductions could be achieved by deploying the same workload on a (potentially cheaper) platform with more modest physical characteristics. Second, spare capacities could be used to accommodate additional workload, so as to enhance existing functionalities, or implement new, more sophisticated ones.

Contribution: A heuristic-based method for deriving static paths of traffic flows at design-time is proposed. When compared with conventional dimension-ordered routing algorithms, our approach succeeds to avoid missed deadlines for a significantly higher fraction of analysed flow-sets, and at the same time conserves the deadlock-free property (**Experiment 1**). A slight decrease in hardware requirements (the number of needed virtual channels) is also achieved (**Experiment 2**). Moreover, the proposed technique finds optimal routes in 23% of the cases, while in more than half of the cases it underperforms by at most 10% with respect to optimal routes (**Experiments 3-4**). The method is computationally efficient and scalable (**Experiment 5**).

II. Related Work

In this section, we describe only studies related to deriving real-time guarantees for priority-preemptive NoCs. Song et al. [9] proposed to use virtual channels as a means to implement preemptions among traffic flows. Shi and Burns [10] further developed that idea, made two additional assumptions (per-traffic-flow distinctive priorities, and per-priority virtual channels), and proposed the worst-case analysis for wormhole-switched priority-preemptive NoCs with fixed priorities and constrained deadlines. Then, Shi et al. [14] extended the model so as to make it applicable to flow-sets with arbitrary deadlines, while Nikolić et al. [15] reduced hardware requirements of this model. Also working on this topic, Shi and Burns [16] and Liu et al. [17] derived methods for priority assignment, while Nikolić and Petters [18] introduced a novel arbitration policy and the accompanying worst-case analysis for flows with dynamic priorities. Kashif et al. [19] proposed a *stage level analysis* which performs better than the traditional analysis, but requires sufficiently big buffers to store preempted flows.

Researchers also used different mapping strategies, with the main objective to accommodate more traffic flows on priority-preemptive NoCs, without missing deadlines. Shi and Burns [20] perform a simple task swapping, Mesidis and Indrusiak [21] and Racu and Indrusiak [22] use generic algorithms, while Nikolić et al. [15] employ the simulated annealing metaheuristic. Sayuti and Indrusiak [23] studied the mapping process, with an emphasis on the NoC power consumption.

Assuming the same type of NoC, Indrusiak [24] proposed an end-to-end schedulability analysis for a fully-partitioned many-core system, while Nikolić et al. [25] derived a communication analysis for Limited Migrative Model. Burns et al. [26] and Indrusiak et al. [27] demonstrated that priority-preemptive NoCs can also be used in the mixed-criticality domain, which is the area that currently attracts a lot of attention from the real-time community.

The aforementioned studies assume static routing, and the X-Y routing policy was used in experiments. Yet, as mentioned in Section I, that approach has some limitations. Motivated by those facts, in this work we focus on deriving flow-paths that do not necessarily conform to the rules of the said policy, and investigate to what extent can such an approach contribute to a more efficient use of NoCs in the hard real-time domain.

III. System Model

A. Platform

The platform under consideration θ is a many-core system with a 2-D mesh NoC interconnect medium. The NoC consists of $m \times n$ identical routers (like illustrated in Figure 1). Each pair of adjacent routers is connected with two unidirectional links, and all links have the same physical characteristics. The data transfer employs the wormhole switching mechanism with the credit-based flow control. With this technique, the data is, prior to sending, divided into small elements of fixed size called *flits*. All flits that constitute one traffic packet are sequentially injected into a NoC, and then travel to their destination in a pipeline manner.

A virtual channel (VC) is a buffer dedicated to a specific traffic flow in a specific port. As mentioned before, VCs can be used as an infrastructure to implement preemptions among flows. For example, when two flows contend for a common link, one will be granted the permission to progress (the higher-priority one), while the other one will be stored inside a VC, until the NoC resources of interest become available for its traversal. As in all related works (Section II), we assume that the number of VCs is at least equal to the maximum number of contentions for any port [15], which guarantees that each flow will have an available VC within each router along its path. This assumption is realistic, because on a NoC with a 10×10 2-D mesh, the workload of 400 flows requires on average only 10 VCs [15]. Moreover, we assume that each VC can store only a single flit.

Additionally, we assume that the routing of flows is not conforming to some global policy (e.g. X-Y). In fact, each flow will have its static path derived at design-time (the contribution of this work) and the path information will be stored inside the header flit. Once a packet is released, intermediate routers will inspect its header flit, and route the packet based on the given information. Some of the existing platforms already support this type of routing, e.g. [3]. This topic will be revisited in Section VI-A.

B. Workload

The workload is modelled as a sporadic traffic flow-set $\mathcal{F}$, which is a collection of z flows $\{\phi_1, ..., \phi_z\}$. Each flow ϕ_i is characterised by a source router $\rho_{src}(\phi_i)$, a destination router $\rho_{dst}(\phi_i)$, a minimum inter-arrival period $T(\phi_i)$, a constrained deadline $D(\phi_i) \leq T(\phi_i)$, and a unique fixed priority $P(\phi_i)$[1]. A flow ϕ_i travels across the minimal distance between $\rho_{src}(\phi_i)$ and $\rho_{dst}(\phi_i)$, also known as the *Manhattan distance*, formally introduced in Section VI-A. The term $C(\phi_i)$ denotes the no-load latency of ϕ_i (no contentions), in the literature also referred to as the *basic network latency*. $J_R(\phi_i)$ denotes the release jitter, which is the maximum deviation of two successive packets of ϕ_i released from their period $T(\phi_i)$.

During each inter-arrival period, a flow releases exactly one packet. If it can be analytically proven that each packet of a flow can complete the transfer before its deadline, even

[1] In this work, we assume that priorities are not given *a priori*, as a part of the problem instance, but instead, once all flow-paths are derived, some technique for priority assignment (e.g. [16], [17]) will be applied. Note, that our approach is not tied to any particular method.

in the *worst-case conditions*, then that flow is considered *schedulable*. Let $R(\phi_i)$ denote the worst-case traversal time (WCTT) of packets of ϕ_i. Then $R(\phi_i) \leq D(\phi_i)$ is both a sufficient and necessary condition for ϕ_i to be schedulable. If all flows of the flow-set are schedulable, the flow-set itself is schedulable.

IV. Problem Formulation

Given the platform θ, the flow-set $\mathcal{F}$, and a technique for priority assignment (e.g. [16], [17]), for each flow $\phi_i \in \mathcal{F}$ derive at design-time a static path $\pi(\phi_i)$, such that $\mathcal{F}$ is schedulable. A path $\pi(\phi_i)$ denotes a set of resources (NoC links of θ) traversed by ϕ_i, from $\rho_{src}(\phi_i)$ to $\rho_{dst}(\phi_i)$.

V. Background and Preliminaries

In this section, we cover the real-time analysis of wormhole-switched priority-preemptive NoCs with fixed priorities.

Let $\mathcal{F}_D(\phi_i)$ be a set of flows that can preempt ϕ_i (share some NoC links with ϕ_i and have higher priorities). Formally:

$$\forall \phi_j \in \mathcal{F} \mid P(\phi_j) > P(\phi_i) \wedge \pi(\phi_j) \cap \pi(\phi_i) \neq \emptyset \Rightarrow \phi_j \in \mathcal{F}_D(\phi_i)$$

The WCTT of ϕ_i is computed by solving Equation 1 [10].

$$R(\phi_i) = C(\phi_i) + \sum_{\forall \phi_j \in \mathcal{F}_D(\phi_i)} \left\lceil \frac{J_R(\phi_j) + R(\phi_i) + J_I(\phi_j)}{T(\phi_j)} \right\rceil \cdot C(\phi_j) \quad (1)$$

In Equation 1, the term $J_I(\phi_j)$ is the interference jitter, and it is computed by solving Equation 2.

$$J_I(\phi_j) = \begin{cases} R(\phi_j) - C(\phi_j), & \text{if } \exists \phi_k \in \mathcal{F}_D(\phi_j) \mid \phi_k \notin \mathcal{F}_D(\phi_i) \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

That is, the interference jitter has a non-zero value only if there exists a higher-priority flow ϕ_k, which is not able to preempt the flow under analysis ϕ_i, but is able to preempt some other flow ϕ_j, which can preempt ϕ_i. This effect is also known as the *indirect interference*. It is evident that in Figure 1 there is no indirect interference, because there is no flow which satisfies the condition of Equation 2. However, that may not be the case with the example from Figure 2. Let us consider that example, and let flow indexes also be the priorities. Then, ϕ_2 suffers indirect interference from ϕ_4, which is manifested in its WCTT as an interference jitter of ϕ_3. Note, that in order to compute the WCTT of ϕ_i, it is necessary to have the WCTTs already computed for all flows from $\mathcal{F}_D(\phi_i)$. Also note, that the priority order among flows from $\mathcal{F}_D(\phi_i)$ does matter.

VI. Proposed Approach

The proposed path derivation method is presented as follows. First, we estimate the search space of the addressed problem, and justify the need for the heuristic-based approach (Section VI-A). Then, we introduce the metric for the path derivation (Section VI-B), and demonstrate how it is used to derive a path of a single flow (Section VI-C). Finally, we show how to derive paths of an entire flow-set (Section VI-D).

A. Search Space Reduction via Minimal Paths

In this section, we estimate the search space of the path derivation problem, so as to asses whether an exhaustive search is a viable option. First, we formally introduce the concept of the *minimal path*.

Definition 1 (Minimal path)**.** *A minimal path of the flow ϕ_i, between the source router $\rho_{src}(\phi_i)$ and the destination router $\rho_{dst}(\phi_i)$, with the coordinates $[x_{src}(\phi_i); y_{src}(\phi_i)]$, and $[x_{dst}(\phi_i); y_{dst}(\phi_i)]$, respectively, is any continuous path which involves $|x_{src}(\phi_i) - x_{dst}(\phi_i)| + |y_{src}(\phi_i) - y_{dst}(\phi_i)| + 1$ routers, of which $\rho_{src}(\phi_i)$ is the first, and $\rho_{dst}(\phi_i)$ is the last.*

In this work, we impose a constraint that each derived path must be minimal. Note that some flows may have multiple minimal paths (e.g. Figure 1). Notice that, by design, the X-Y routing method derives minimal paths. The benefits when considering only minimal paths are as follows:

• **Constant no-load latencies:** Regardless of selected paths, no-load latencies of all flows are constant, i.e. $C(\phi_i) = const, \forall \phi_i \in \mathcal{F}$. Therefore, no recalculations of no-load latencies are necessary when deriving or modifying flow-paths.

• **Solution space reduction:** The solution space is reduced to more promising candidates. That is, if a flow is unschedulable with any of its minimal paths, it is unlikely that it will be schedulable with any other path. We back up this claim with the fact that non-minimal paths involve more shared resources, and therefore it is more likely that a flow will face more interference on a non-minimal path, than on a minimal one.

• **Path can be stored in the header as a set of bits:** Assuming the X-Y routing policy, routing decisions are based on a destination router, whose information is stored in a header flit. Yet, in our approach it is necessary to store a path information as well. When considering minimal paths only, a path can be represented as a set of bits, with a bit per router, except the last one, where it is not needed. That is, solely by analysing the information regarding the destination, the current router can deduce the horizontal and/or vertical direction in which the flow has to progress. The only remaining thing is to decide which one to chose. So, if we establish a convention that "0" means a horizontal advancement, and "1" means a vertical progress, then the path of a flow can be described as a set of bits. With this convention, the path of ϕ_1 from Figure 1a is $\{0, 0, 0, 1, 1, 1\}$, while the path of ϕ_1 from Figure 1b is $\{1, 1, 0, 1, 0, 0\}$. Thus, all routing information can be stored in a header flit, and each traversed router will route a packet by (i) considering the final destination, (ii) interpreting the corresponding bit, and (iii) performing a logical shift of the path information inside the header (so as to set the corresponding bit for the next router). On Kalray [3], traffic is routed in a similar way[2].

• **Minimal paths cannot cause deadlocks:** Lemma 1 proves that a flow with a minimal path cannot put itself in a deadlock, while Theorem 1 proves that a flow-set with all minimal paths cannot experience a deadlock.

Lemma 1. *On a wormhole-switched priority-preemptive NoC, a flow ϕ_i, with a minimal path $\pi(\phi_i)$, cannot deadlock itself.*

[2]On Kalray [3], two bits are used per router, and each of the four progress directions has its own explicit code $(00, 01, 10, 11)$.

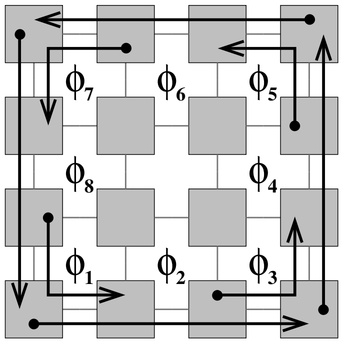


Fig. 2: Example of flows with minimal paths

Proof: Proven by contradiction. Assume the opposite, that the flow ϕ_i can deadlock itself. This implies that ϕ_i competes with itself for some NoC resource, e.g. a link λ. In order to do that, ϕ_i has to traverse λ twice, and hence λ exists twice in $\pi(\phi_i)$. This means that $\pi(\phi_i)$ is not minimal, which contradicts the initial assumption. A contradiction has been reached. ■

Theorem 1. *Assuming a priority-preemptive NoC with the wormhole-switching, a flow-set with all minimal paths cannot experience a deadlock.*

Proof: Proven by contradiction. Assume the opposite, that a set of interfering flows reached a deadlock. Let ϕ_i be the flow with the lowest priority. By the initial assumption, it cannot progress because it is contending with some higher-priority flow ϕ_j. Also, ϕ_j cannot progress due to some higher-priority flow ϕ_k. Let us continue this reasoning, until we encounter ϕ_w, which is the highest priority flow involved in a deadlock (e.g. if we assume that the flows from Figure 2 reached a deadlock, and that flow indexes are also the respective priorities, then ϕ_8 would be the flow with the highest priority). By the initial deadlock assumption, ϕ_w cannot progress. Since there exists no higher-priority flow which might prevent its progress, this means that ϕ_w put itself in a deadlock. This contradicts Lemma 1. A contradiction has been reached. ■

After narrowing a set of possible paths to minimal only, and after justifying such an approach, let us estimate the size of the reduced solution space. This will give us a good indication whether it is practical to perform an exhaustive enumeration. Let $\Pi(\phi_i)$ be a set of all possible minimal paths of a given flow ϕ_i. Theorem 2 computes the number of elements of $\Pi(\phi_i)$.

Theorem 2. *Let ϕ_i be a flow between the source router $\rho_{src}(\phi_i)$ and the destination router $\rho_{dst}(\phi_i)$, with the coordinates $[x_{src}(\phi_i); y_{src}(\phi_i)]$, and $[x_{dst}(\phi_i); y_{dst}(\phi_i)]$, respectively. Moreover, let: (i) h_i be the horizontal distance, (ii) v_i be the vertical distance, and (iii) s_i be the minimal (Manhattan) distance between $\rho_{src}(\phi_i)$ and $\rho_{dst}(\phi_i)$.*

$$h_i = |x_{src}(\phi_i) - x_{dst}(\phi_i)|$$

$$v_i = |y_{src}(\phi_i) - y_{dst}(\phi_i)|$$

$$s_i = h_i + v_i = |x_{src}(\phi_i) - x_{dst}(\phi_i)| + |y_{src}(\phi_i) - y_{dst}(\phi_i)|$$

The number of minimal paths of ϕ_i, denoted by $E(\phi_i)$, is:

$$E(\phi_i) = \binom{s_i}{h_i} = \binom{s_i}{v_i} = \frac{s_i!}{h_i! \cdot v_i!} \tag{3}$$

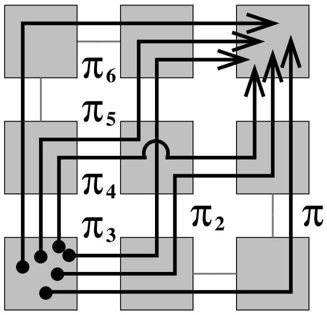


Fig. 3: Minimal paths between diagonal routers on 3×3 mesh

Proof: Proven directly. Recall the proposed convention for storing the path information into the header flit as a set of bits ("0" for a horizontal advancement, and "1" for a vertical progress). A bit is needed for each traversed router, except the last one (which is the destination), thus the total number of bits needed to describe an entire path is s_i. Regardless of the selected path, the flow must progress h_i times horizontally, and v_i times vertically, which implies that of those s_i bits, h_i must be zeros and v_i must be ones. Therefore, the problem becomes to find the total number of different s_i-bit-long binary sequences with h_i zeros and v_i ones. Note that the relative order of the same-value bits does not matter, and therefore the total number of possible solutions is equal to the number of permutations of all s_i bits ($s_i!$), divided by the number of permutations of ones ($h_i!$) and the number of permutations of zeros ($v_i!$). That is:

$$E(\phi_i) = \frac{s_i!}{h_i! \cdot v_i!} = \binom{s_i}{h_i} = \binom{s_i}{v_i}$$

■

This can be easily verified with an illustrative example given in Figure 3, where the source router of the flow ϕ_i is in the bottom-left corner, and the destination router is in the top-right corner of a 3×3 mesh NoC. Let us compute the total number of possible minimal paths of ϕ_i. In this example, $s_i = 4$, $h_i = 2$ and $v_i = 2$. Therefore:

$$E(\phi_i) = \binom{4}{2} = \frac{4!}{2! \cdot 2!} = 6$$

And indeed, from Figure 3 it is visible that the total number of possible minimal paths of ϕ_i is 6.

The parameter $E(\phi_i)$ we call the *elasticity property* of ϕ_i. Notice two things: (i) flows which connect diagonal corners of the grid have the biggest elasticity property and (ii) flows which connect routers from the same row or column have the smallest elasticity property (equal to one). Indeed, for a flow ϕ_i from the latter category it holds that $h_i = 0 \lor v_i = 0 \Rightarrow E(\phi_i) = 1$, and hence ϕ_i has only a single minimal path. The elasticity property will be used during the path derivation process (revisited in Section VI-D).

Now we can estimate the search space of the problem formulated in Section IV, with an additional restriction that paths must be minimal. Let $X(\mathcal{F})$ be the size of the solution space. Then, $X(\mathcal{F})$ is computed by solving Equation 4.

$$X(\mathcal{F}) = \prod_{\forall \phi_i \in \mathcal{F}} E(\phi_i) \tag{4}$$

Despite the search space reduction (only minimal paths are considered), an exhaustive search for the optimal solution is, in most cases, still impractical. For example, consider only 15 flows on a 8×8 NoC, and assume that each flow has the elasticity property equal to 10, which is a reasonable assumption for the NoC of that size. Then, $X(\mathcal{F}) = 10^{15}$, and we believe that this result justifies a heuristic-based approach for solving the path derivation problem. Therefore, in the next section we introduce a new concept, which is used as a heuristic function in our method.

B. *Indicative Traversal Time (ITT)*

Although in the end it will be necessary to compute the WCTT of each flow, so as to validate that the flow-set is indeed schedulable with derived paths, using the WCTT analysis (Equation 1) during the path derivation phase may not be the most efficient approach. This claim is supported by the fact that in order to compute the WCTT of a flow, it is necessary to already have computed WCTTs of all higher-priority directly interfering flows. In other words, the analysis holds only for given paths and only for a given priority ordering among flows (recall Section V). This implies that each time the WCTT analysis is applied, only an individual candidate, or a small subset of related candidates, of an entire solution space is assessed.

Given the nature of the problem and the size of the search space, a more efficient approach would be to apply some generic metric which allows to systematically asses *classes of candidates* (potential solutions), and in that way converge towards a set of promising ones. To that aim, we introduce the *indicative traversal time* (ITT). Before we explain how to compute the ITT of a given flow, let us define the set of flows $\mathcal{F}_A(\phi_i)$, which contains all flows that share the path with the analysed flow ϕ_i, regardless of their priorities. Formally:

$$\forall \phi_j \in \mathcal{F} \mid j \neq i \wedge \pi(\phi_j) \cap \pi(\phi_i) \neq \emptyset \Rightarrow \phi_j \in \mathcal{F}_A(\phi_i)$$

The flow ϕ_i can contend with flows from $\mathcal{F}_A(\phi_i)$. Which of those can preempt ϕ_i, and which can be preempted by it, depends on the priority assignment, and is irrelevant at this stage. The ITT of ϕ_i is computed by solving Equation 5.

$$R^*(\phi_i) = C(\phi_i) + \sum_{\forall \phi_j \in \mathcal{F}_A(\phi_i)} \left\lceil \frac{J_R(\phi_j) + R^*(\phi_i)}{T(\phi_j)} \right\rceil \cdot C(\phi_j) \quad (5)$$

Notice the similarities between Equation 5 and Equation 1. In fact, there are only two differences between the WCTT and ITT. First, the ITT considers all contending flows, regardless of their priorities, whereas WCTT takes into account only higher-priority contending flows. Second, Equation 5 does not have an interference jitter component, and the ITT of the analysed flow does not depend on the WCTT, nor the ITT of other flows.

There are several benefits of using the ITT:

• **Easy to compute:** As already mentioned, the ITT does not require its components to be pre-computed, but only depends on values (constants), which are all given as inputs.

• **Generic:** The ITT does not depend on the priority ordering among traffic flows, but only on their paths.

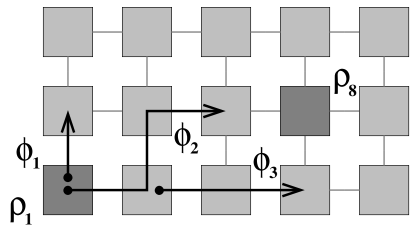


Fig. 4: Example of flows to explain the process of finding the path with the smallest ITT for the flow ϕ_4 from ρ_1 to ρ_8

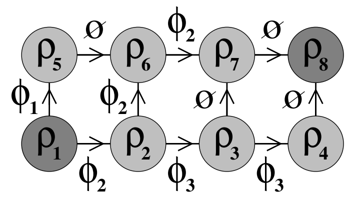


Fig. 5: Direct acyclic graph for Figure 4

• **Good indication:** The ITT gives a good indication about the level of contention among flows on the path of the analysed flow, where a small value means that a path is low utilised.

The intuition behind our approach is to use the ITT metric to quickly compare multiple paths (where exist), and for each flow select the one which would most likely lead to a schedulable solution, if one exists. It is a reasonable assumption that the most promising path is the one with the smallest ITT, and our approach is developed around this idea. Note, that from the ITT nothing can be inferred about the WCTT, nor the schedulability of the analysed flow, simply because the ITT is oblivious to priorities. The sole purpose of this metric is to help derive a path of each flow, so that after employing some priority assignment strategy (e.g. [16], [17]), a flow-set is likely to be rendered schedulable. In the next section, we describe the process of finding a path with the smallest ITT.

C. *Finding Path with the Smallest ITT*

The process is formulated with Algorithm 1. We also use a small demonstrative example of flows (Figure 4), so as to help the reader to better understand the proposed method.

Consider the set of flows from Figure 4, and let ϕ_4 from ρ_1 to ρ_8 be the flow for which we want to find a path with the smallest ITT. First, a direct acyclic graph (DAG) is created: routers are represented as vertices, and links are represented as edges. The weight of an edge is a set of flows that traverse it. Note that the entire NoC needs not be considered, but only (i) routers whose x- and y-coordinates are between $[min\{x_{src}(\phi_i), x_{dst}(\phi_i)\}; max\{x_{src}(\phi_i), x_{dst}(\phi_i)\}]$ and $[min\{y_{src}(\phi_i), y_{dst}(\phi_i)\}; \quad max\{y_{src}(\phi_i), y_{dst}(\phi_i)\}]$, (ii) horizontal links in the direction from $x_{src}(\phi_i)$ to $x_{dst}(\phi_i)$, and (iii) vertical links in the direction from $y_{src}(\phi_i)$ to $y_{dst}(\phi_i)$. The justification is that minimal paths traverse only these routers and only these links. A corresponding DAG for the example from Figure 4 is given in Figure 5.

After constructing a DAG (line 1 of Algorithm 1), a single path is created and added to the set of all paths (lines $2-3$). A path contains only a source router, and its ITT is set to the no-load latency of the flow – $C(\phi_i)$. In order to control the complexity of the approach, the variable *steps* is introduced,

Algorithm 1 $findPathWithSmallestITT(\theta, \mathcal{F}, \phi_i)$

```
Input: platform $\theta$, flow-set $\mathcal{F}$, analysed flow $\phi_i$
Parameters: maximum number of allowed steps $max\_steps$
Output: path of $\phi_i$ with the smallest ITT
 1: $dag.Create(\theta, \mathcal{F}, \phi_i)$; // create DAG for $\phi_i$
 2: $paths.Initialise()$; // create empty set of paths
 3: $paths.Add(path(\rho_{src}(\phi_i), C(\phi_i)))$; // first path only $\rho_{src}(\phi_i)$
 4: $steps \leftarrow 1$; // initialise number of steps
 5: while $(true)$ do
 6:     // find and remove path with the smallest ITT
 7:     $minPath \leftarrow paths.GetMinPath()$;
 8:     if $(minPath.dst = \rho_{dst}(\phi_i))$ then
 9:         // path with the smallest ITT has been found
10:         return $minPath$;
11:     end if
12:     if $(steps = max\_steps)$ then
13:         // maximum number of steps reached
14:         $currPath \leftarrow paths.GetMinPathWithDst(\rho_{dst}(\phi_i))$;
15:         if $(currPath \neq null)$ then
16:             // path found, but it may not have the smallest ITT
17:             return $currPath$;
18:         else
19:             // $\rho_{dst}(\phi_i)$ not reached, so return X-Y routed path
20:             return $XYpath(\phi_i)$;
21:         end if
22:     end if
23:     $newPaths \leftarrow minPath.Develop()$; // develop paths
24:     for each $(newPath$ in $newPaths)$ do
25:         $newPath.ComputeITT()$;
26:         $paths.Add(newPath)$; // add new paths to the list
27:     end for
28:     $steps \leftarrow steps + 1$; // increment number of steps
29: end while
```

and given an initial value of 1 (line 4). Then, the path with the smallest ITT is removed from the list (line 7). If its destination is the same as the destination of the analysed flow, then the path with the smallest ITT has been found (lines $8 - 11$). Otherwise, it is checked if the maximum number of steps is exceeded, where *max_steps* is an input parameter which controls how many search iterations (steps) are allowed. A higher value of this parameter means that more computational resources can be spent in the searching phase. If the limit on the number of steps is exceeded, and the currently selected path with the smallest ITT does not have the same destination as the analysed flow, it is checked if some other path exists in the *paths* set, such that it's destination is the same as that of the analysed flow. Note, that for such a path, if it exists, cannot be guaranteed that it has the smallest ITT among all minimal paths. However, since the search process cannot continue further (*max_steps* reached), that path is returned (lines $15 - 17$). Otherwise, if such a path does not exist, an X-Y-routed path is returned (lines $18 - 21$).

If the number of steps did not exceed the limit, a path with the smallest ITT is developed in all possible directions, and new paths are created (line 23). In our example, an initial path $\{\rho_1\}$ is developed into: $\{\rho_1, \rho_2\}$ and $\{\rho_1, \rho_5\}$. Then, for each of the newly created paths, the ITT is computed[3], and the path is added to the list of all paths (lines $24 - 27$). Finally, the number of steps is incremented (line 28).

[3]The no-load latency $C(\phi_i)$ is used in the ITT computation of each analysed path, regardless of its size (e.g. see the first step of Table II).

Assuming the flow parameters given in Table I for our example illustrated in Figures 4-5, the path of the flow ϕ_4 with the smallest ITT is $\{\rho_1, \rho_2, \rho_6, \rho_7, \rho_8\}$, its ITT is 20, and it is found in 7 steps. The step-by-step computation process is given in Table II. In each step, a path with the smallest ITT (the one to be developed) is emphasised with a gray colour.

TABLE I: Flow parameters for Figure 4

Flow	C	J_R	D	T
ϕ_1	5	0	100	100
ϕ_2	10	0	100	100
ϕ_3	20	0	100	100
ϕ_4	10	0	100	100

TABLE II: Process of finding the path of the flow ϕ_4 from ρ_1 to ρ_8 (Figure 4, and Table I) with the smallest ITT value

Step	Path	ITT
1	$\{\rho_1\}$	10
2	$\{\rho_1, \rho_2\}$	20
	$\{\rho_1, \rho_5\}$	15
3	$\{\rho_1, \rho_2\}$	20
	$\{\rho_1, \rho_5, \rho_6\}$	15
4	$\{\rho_1, \rho_2\}$	20
	$\{\rho_1, \rho_5, \rho_6, \rho_7\}$	25
5	$\{\rho_1, \rho_2, \rho_3\}$	40
	$\{\rho_1, \rho_2, \rho_6\}$	20
	$\{\rho_1, \rho_5, \rho_6, \rho_7\}$	25
6	$\{\rho_1, \rho_2, \rho_3\}$	40
	$\{\rho_1, \rho_2, \rho_6, \rho_7\}$	20
	$\{\rho_1, \rho_5, \rho_6, \rho_7\}$	25
7	$\{\rho_1, \rho_2, \rho_3\}$	40
	$\{\rho_1, \rho_2, \rho_6, \rho_7, \rho_8\}$	20
	$\{\rho_1, \rho_5, \rho_6, \rho_7\}$	25

For clarity of the example, paths are represented as sets of routers, whereas paths are usually represented as sets of traversed links. The link representation is more convenient, because having the common router is only a necessary condition for the contention between two flows, while having a common link is both a necessary and sufficient condition.

A reader may notice that the proposed technique is similar to Dijkstra's algorithm [28] for finding the shortest path between two graph nodes. Yet, unlike Dijkstra's algorithm, which can be applied to weighted DAGs with the complexity of $O(E+V)$ (E being the number of edges, and V being the number of vertices), our approach may require, in the worst-case, to investigate all possible paths. The difference comes from the fact that, in Dijkstra's algorithm, a local minimum will always lead to the global minimum, whereas that is not the case for our problem. We illustrate this with an example from Figure 5 and the corresponding ITT computation process from Table II. Notice that in *step 3* there is a path $\{\rho_1, \rho_5, \rho_6\}$, while in *step 5* there is a path $\{\rho_1, \rho_2, \rho_6\}$. Both paths lead to ρ_6, but with different ITTs, which is in the former case 15, and in the latter 20. However, despite having a smaller ITT, the former path *cannot* suppress the latter one, because it may happen that the latter path eventually leads to the one with the smallest ITT. And in fact that happens in our example, because the path with the smallest ITT is $\{\rho_1, \rho_2, \rho_6, \rho_7, \rho_8\}$. So, none of the paths can be discarded until a path is found, such that (i) it has the desired destination, and (ii) it has the smallest ITT among all currently investigated paths. And due to these reasons, we introduced the variable *max_steps*, which represents a means to control the algorithm complexity. In Section VII, we will investigate its usefulness.

D. Path Derivation

The process of deriving flow-paths is given in Algorithm 2. Recall the elasticity property, which represents the number of possible minimal paths of a given flow. First, by solving Equation 3 for each flow, we compute their respective elasticity properties (line 2). Then, we initialize a path of each flow to an empty set (line 3). We mentioned in Section VI-A that flows whose source and destination routers are in the same row or column have a single minimal path. All single-path flows are added to the set *spFlows* (lines $6-8$). Remaining flows are added to the set of multi-path flows *mpFlows* (lines $4-5$).

Each flow from *spFlows* is assigned its only minimal path (lines $10-12$). In order to exploit the elasticity property of multi-path flows more efficiently, we sort them by the parameter E, non-decreasingly (line 13), and treat them in that order. The intuition behind this approach is to assign paths first to flows with fewer choices, and thus allow flows which have numerous options to select their respective paths later, when the contentions are more apparent. Similar to Algorithm 1, there is a parameter which controls the algorithm complexity, called LIM. This parameter serves as an upper bound on the maximum number of iterative steps that can be performed when deriving paths. Now, for each flow from *mpFlows*, a path which was derived in the previous iteration (or an empty path, if it is the first iteration) is stored as an old path (line 16). Then, a new path is derived by invoking Algorithm 1. This is repeated until all flows from *mpFlows* are considered. If paths of all flows are the same for two consecutive iterations, the algorithm stops (lines $19-21$), and a flow-set is rendered unschedulable (line 27). Otherwise, a flow-set passes through a priority assignment phase (line 22), by using some priority assignment method (e.g. [16], [17]).

Now, the flow-set is tested for schedulability. If the outcome is positive, the algorithm returns $true$ (lines $23-25$). Otherwise, a new iteration begins. In the new iteration, all flows from *mpFlows* have another chance to chose their paths, but unlike in the first iteration, this time all flows except the currently analysed one will already have their respective paths derived. This allows to re-align flows on the NoC, and achieve even better arrangements than in the previous iteration(s). Then, if a priority assignment can be performed with new paths, such that a flow-set is schedulable, the algorithm is terminated with the positive reply (lines $23-25$). Otherwise, if new paths are the same as in the previous iteration when a flow-set was unschedulable, then no further improvements can be made, and the algorithm terminates with a negative reply (lines $19-21$ and 27). Finally, if a flow-set cannot be made schedulable even after LIM number of iterations, the algorithm terminates with a negative reply (lines 14 and 27).

Algorithm 2 $assignSchedulablePathsAndPriorities(\theta, \mathcal{F})$

```
Input: platform $\theta$, flow-set $\mathcal{F}$
Parameters: max. number of allowed iterations $LIM$
Output: info. whether flow-set is schedulable or not
 1: for each ($\phi_i$ in $\mathcal{F}$) do
 2:     $E(\phi_i) \leftarrow CompElasticity(\phi_i)$; // Equation 3
 3:     $\pi(\phi_i) \leftarrow \emptyset$; // Set empty paths for all flows
 4:     if ($E(\phi_i) > 1$) then
 5:         $mpFlows.Add(\phi_i)$; // added to set of multi-path flows
 6:     else
 7:         $spFlows.Add(\phi_i)$; // added to set of single-path flows
 8:     end if
 9: end for
10: for each ($\phi_i$ in $spFlows$) do
11:     $\pi(\phi_i) \leftarrow FindMinPath(\phi_i)$; // $\phi_i$ has one minimal path
12: end for
13: $mpFlows.Sort(E\uparrow)$; // sort by $E$, non-decreasingly
14: for ($i \leftarrow 0; i < LIM; i \leftarrow i+1$) do
15:     for each ($\phi_i$ in $mpFlows$) do
16:         $\pi_{old}(\phi_i) \leftarrow \pi(\phi_i)$;
17:         $\pi(\phi_i) \leftarrow findPathWithSmallestITT(\theta, \mathcal{F}, \phi_i)$; // invoke Algorithm 1
18:     end for
19:     if ($\nexists \phi_i \in spFlows \mid \pi_{old}(\phi_i) \neq \pi(\phi_i)$) then
20:         break;
21:     end if
22:     $PriorityAssignment(\mathcal{F})$;
23:     if ($IsSchedulable(\mathcal{F})$) then
24:         return $true$;
25:     end if
26: end for
27: return $false$;
```

VII. Experimental Evaluations

In this section, we show the results of the experimental evaluation. The emphasis is on 4 aspects: (i) schedulability guarantees (**Experiment 1**), (ii) hardware requirements (**Experiment 2**), (iii) the method efficiency (**Experiments 3-4**), and (iv) the computational aspects and scalability (**Experiment 5**). The following approaches are compared:

• our method, as presented in Section VI-D.

• the same approach, with the only difference that in line 17 of Algorithm 2, instead of invoking Algorithm 1 to find a path with the smallest ITT, a classical Dijkstra's shortest path algorithm is used. A weight of each edge e_i is interpreted as a cumulative utilisation of all flows traversing a corresponding link λ_i, i.e. $w(e_i) = \sum\limits_{\forall \phi_j \in \mathcal{F} \mid \lambda_i \in \pi(\phi_j)} \frac{C(\phi_i)}{T(\phi_i)}$.

• a method where paths conform to the X-Y routing policy.

• a method where paths conform to the Y-X routing policy.

To make the evaluation fair, for all approaches we use the same priority assignment method – a heuristic function *H6* (Equation 11 from the work of Shi and Burns [16]), without backtracking[4]. We opt for this method because it is very fast and efficient, and hence gives us the opportunity to devote the most of the experimentation time to the most important aspect of this work – the assessment of routing techniques.

Evaluation Parameters

The analysis parameters are given in Table III. An asterisk sign denotes a randomly generated value (uniform distribu-

[4]Performing evaluations where our method exploits more time-consuming priority assignment schemes (e.g. [16], [17]) is a potential future work.

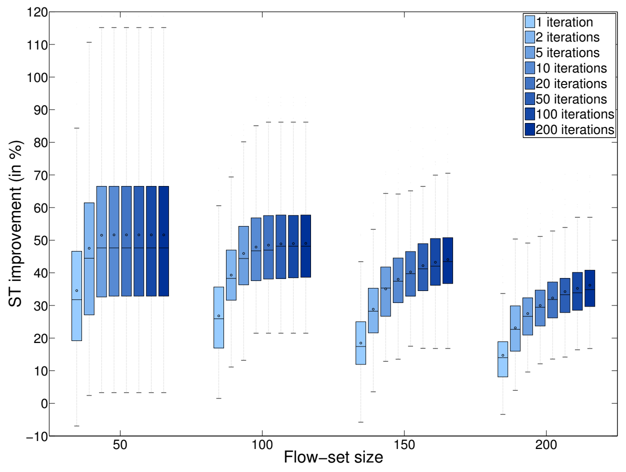


Fig. 6: ST improvements our over $best\{XY, YX\}$

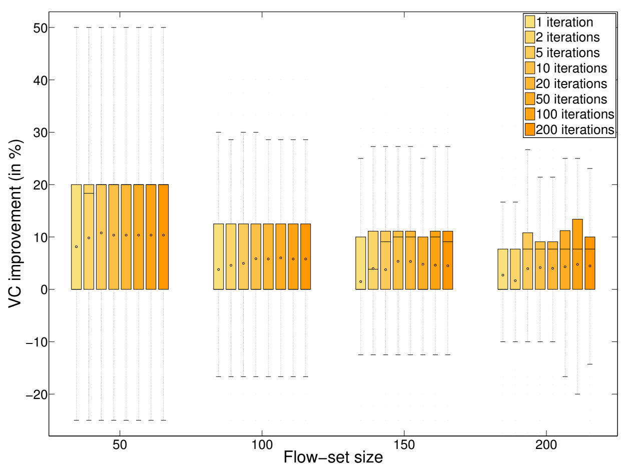


Fig. 7: VC improvements our over $best\{XY, YX\}$

tion). The no-load latencies (Cs) are obtained, from platform characteristics and flow sizes, in the same way as in the studies of Shi and Burns [10] and Nikolić and Petters [18].

TABLE III: Analysis parameters

NoC topology	**2-D mesh**
Router frequency	**2GHz**
Router latency + link latency	**3 + 1 cycles**
Link width = flit size	**4B**
Flow size	**[1 - 128]* KB**
Flow periods	**[20 - 100]* μs**
max_steps for Algorithm 1	**max{100, 10% $\cdot$ $E(\phi_i)$}**

Experiment 1: Schedulability Guarantees

The comparison of the approaches is performed in the form of a *sensitivity analysis*. Assuming that the method, the platform and the flow-set are already selected, the flow-set is first tested for schedulability with the initial flow sizes. If it is schedulable, the sizes of all flows are uniformly increased until the flow-set becomes unschedulable. If it is unschedulable, the sizes of all flows are uniformly decreased, until the flow-set becomes schedulable. The process is repeated via a binary search, until a threshold is found. We term that value the *schedulability threshold* ST. A bigger ST implies that a method is more efficient.

In this experiment, we focus on the schedulability guarantees, so we compare STs in the following way. Let ST_1 and ST_2 be obtained for a given platform and a given flow-set with two different approaches, and let $ST_1 > ST_2$. Then, the improvement of the former method over the latter one is computed as follows: $imp(ST_1) = \frac{ST_1 - ST_2}{ST_2} \cdot 100\%$.

Assuming an 8×8 NoC, we create 200 flow-sets with 50 flows each. Source and destination routers are randomly chosen. For each flow-set, we obtain and compare STs of all 4 approaches (the vertical axis in Figure 6)[5]. In the first two approaches, the parameter LIM, which represents the maximum number of iterations of Algorithm 2, is additionally varied (the legend in Figure 6). This is repeated for flow-sets with 100, 150 and 200 flows (the horizontal axis in Figure 6).

Our method with Dijkstra's shortest path algorithm exhibits a significantly worse performance than the other 3 methods, and is, therefore, omitted from Figure 6 and the remaining experiments. This finding can be explained with the fact that although the algorithm is finding paths with the smallest cumulative utilisation, and hence minimising the per-link contentions, it is at the same time increasing the total number of *per-path contentions*. Thus, on average, each flow is contending with more flows then in any other approach. This has a very negative effect on Equation 1.

For better clarity, in Figure 6 we plot the improvements of our approach (Algorithm 2) over the better of the two remaining methods with conventional routing policies (X-Y and Y-X), termed $best\{XY, YX\}$. It is evident that our method achieves substantially better results than the other approaches. The improvements are the biggest for low and moderately utilised NoCs (50 and 100 flows), where a thoughtful path derivation can significantly reduce the number of contentions. In such scenarios our method excels, and in some cases outperforms the other ones by more than 100%. For larger flow-sets the gains slightly decrease, but even for sets with 200 flows the improvements are greater than 30%, in more than 3/4 of the cases.

The parameter LIM has a significant effect on the efficiency of the approach. For smaller sets, the final solutions are found within several iterations, and additional ones do not help. For larger sets, the benefits of additional iterations are also noticeable, but the gains are decreasing.

Experiment 2: Hardware Requirements

In this experiment, we observe the improvement of our method over the better of the two remaining methods with conventional routing policies (X-Y and Y-X), with respect to the number of needed virtual channels. The number of needed virtual channels is equal to the maximum number of contentions for any link of the NoC [15]. Thus, the method is considered more efficient if less virtual channels are needed. Let VC_1 and VC_2 be the number of needed virtual channels obtained for two methods, when their respective STs were captured, and let $VC_1 < VC_2$. Then, the improvement of the

[5]In Figures 6-10, the box-edges represent the 25th percentile (q_1) and the 75th percentile (q_3), while every data input more than an interquartile range away from the box (i.e. less than $q_1 - (q_3 - q_1)$, or greater than $q_3 + (q_3 - q_1)$) is considered an outlier.

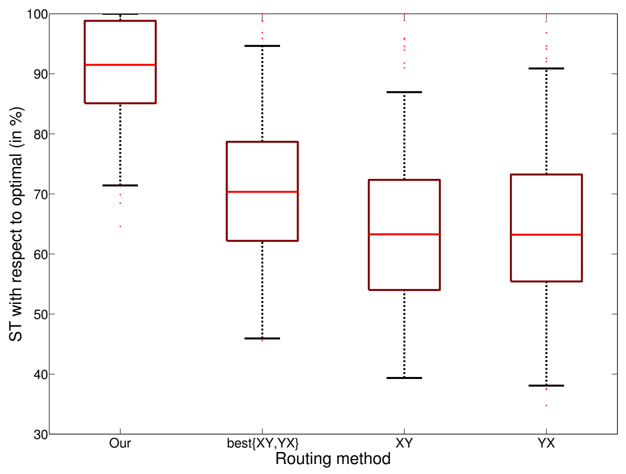


Fig. 8: STs of analysed methods w.r.t. optimal

former method over the latter one is computed as follows: $imp(VC_1) = \frac{VC_2 - VC_1}{VC_2} \cdot 100\%$.

We use the same flow-sets as in the previous experiment, and the results are plotted in Figure 7. In every observed category, our approach reports improvements in $3/4$ of the cases, while it underperforms in the rest of investigated scenarios. For smaller flow-sets the average improvements are between 5 and 10%, while for larger flow-sets the improvements are around 5%. It is visible that additional iterations have a mild positive effect.

Experiments 3 and 4: Method Efficiency

In these two experiments, we test the efficiency of our method. First, we do so by comparing the obtained STs against the corresponding optimal thresholds (obtained by exhaustively enumerating all possible minimal paths and priorities for all flows). Let ST_1 be the threshold obtained by our method for a given flow-set, and let ST_1^{opt} be its corresponding optimal threshold. It always holds that $ST_1 \leq ST_1^{opt}$, and the closer ST_1 to ST_1^{opt} is, the more efficient the method is. This is expressed with the following metric: $eff(ST_1) = \frac{ST_1}{ST_1^{opt}} \cdot 100\%$. In order an optimal search to complete within a reasonable time, we reduced the NoC size to 4×4 and the flow-set size to 30. With these settings, we obtained STs with our approach, with two conventional routing methods (for reference), and optimal thresholds for 200 flow-sets. The results are plotted in Figure 8.

Our method, combined with the fast priority assignment heuristic, manages to find the optimal solution in 23% of the scenarios. The value of the median is 90.46%, which means that, in more than half of the cases, the obtained STs are at most 10% worse than the optimal thresholds. Also, in $3/4$ of the cases, the obtained STs are at most 15% worse than the optimal, which means that in majority of cases the proposed method, combined with the fast priority assignment scheme, finds near-optimal solutions. Figure 8 shows that even the best of the two conventional routing policies performs worse.

In Experiment 4, we analyse the traffic distribution across the NoC. We use the results from Experiment 1 (our method, 100 flows, 200 flow-sets, 200 iterations), and compare the individual traffic of all routers, normalised by the number of

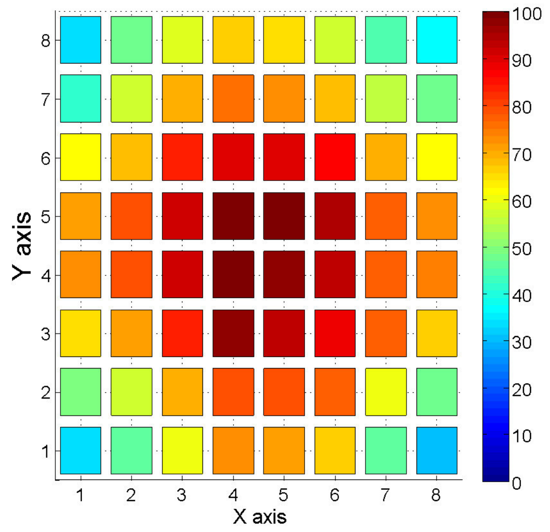


Fig. 9: Traffic distribution across the NoC

corresponding ports. The baseline for comparison is the router with the most traffic (100% in Figure 9).

Figure 9 shows that the most used routers are those in the middle of the NoC, which is a very intuitive result. However, it is evident that even less utilised routers (especially those in the corners) also have a significant amount of traffic, which infers that the method takes advantage of available resources and efficiently distributes the traffic. Extending our approach to also consider non-minimal paths, and combining it with some mapping strategies (e.g. [21], [22]), with an intention to achieve an even better utilisation of NoC resources, is a potential future work.

Experiment 5: Computational Aspects and Scalability

In Section VI-C, it was mentioned that the process of finding the path with the smallest ITT may require, in the worst case, to investigate all possible paths. Investigating all minimal paths of all flows can be computationally demanding. For that purpose, the parameter *max_steps* was introduced.

In this experiment, we investigate how efficient is Algorithm 1 in finding the path with the smallest ITT, and in how many cases did the search process reach the imposed limit on the number of allowed search steps (see the last row in Table III). To that end, we reuse the 200 flow-sets (each with 200 flows) from Experiment 1. For each flow ϕ_i, we measure the number of investigated paths while executing Algorithm 1, and express them as a fraction of the maximum possible value – the elasticity property $E(\phi_i)$ (the vertical axis in Figure 10). We perform that for all flows with different elasticity properties (the horizontal axis in Figure 10). We repeat the process for different NoC sizes, so as to see if the approach scales with the NoC size (see the legend in Figure 10).

To our surprise, the maximum number of steps was not reached in a single case, as is visible from Figure 10. This implies that the proposed method is very efficient and finds the path with the smallest ITT within several steps. This finding suggests that the approach is practical, scalable and applicable to workloads and platforms that are available nowadays.

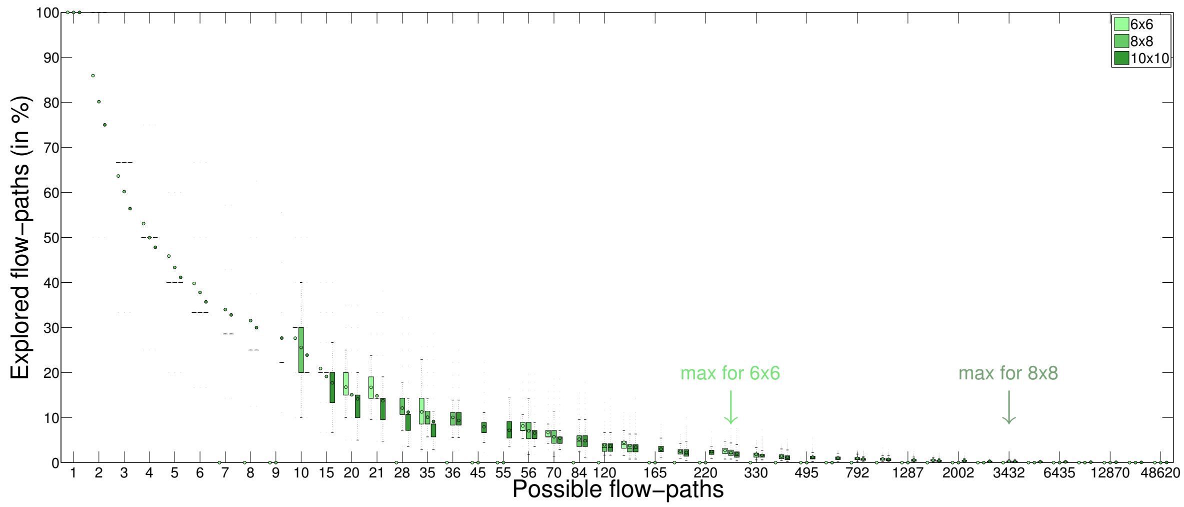


Fig. 10: Number of investigated paths during Algorithm 1, expressed as a fraction of the respective elasticity property

VIII. Conclusions and Future Work

In this work, we studied the impact of the routing flexibility on priority-preemptive NoCs, and their integration in the safety-critical hard real-time domain. We proposed a heuristic-based method for routing traffic flows, and found out that by thoughtfully selecting flow paths, significant improvements can be achieved (with respect to schedulability guarantees) over the approaches which employ traditional routing techniques. At the same time, a slight reduction in hardware requirements is also noticeable. The results of a small-scale experiment suggest that the proposed method, combined with the fast heuristic-based approach for priority assignment, finds optimal or near-optimal routes and priorities in more than 75% of the scenarios.

In order to exploit NoCs even more efficiently, as the future work we plan to develop a unified approach, where our routing method will be combined with mapping techniques (e.g. [15], [21], [22]), and more time-demanding priority assignment schemes (e.g. [16], [17]).

References

[1] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *The Comp. J.*, vol. 35, no. 1, pp. 70 –78, jan 2002.

[2] Intel, *Single-Chip-Cloud Computer*, www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf.

[3] Kalray, *MPPA-256 Manycore Processor*, www.kalray.eu/products/mppa-manycore/mppa-256.

[4] Tilera, *TILE64^{TM} Processor*, www.tilera.com/products/processors/TILEPro_Family.

[5] N. K. Kavaldjiev and G. J. M. Smit, "A survey of efficient on-chip communications for soc," in *4th Symp. Emb. Syst.*, 2003.

[6] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *The Comp. J.*, vol. 26, 1993.

[7] W. Dally, "Virtual-channel flow control," *Trans. Parall. & Distr. Syst.*, vol. 3, no. 2, pp. 194 –205, Mar 1992.

[8] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *Trans. Computers*, 1987.

[9] H. Song, B. Kwon, and H. Yoon, "Throttle and preempt: a new flow control for real-time communications in wormhole networks," in *1997 Int. Conf. Parall. Processing*, Aug 1997.

[10] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *NOCS*, 2008.

[11] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based noc architectures under performance constraints," in *8th ASPDAC*, 2003.

[12] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core cmps," in *36th ISCA*, 2009, pp. 451–461.

[13] B. Nikolić, P. M. Yomsi, and S. M. Petters, "Worst-case memory traffic analysis for many-cores using a limited migrative model," in *19th RTCSA*, 2013.

[14] Z. Shi, A. Burns, and L. S. Indrusiak, "Schedulability analysis for real time on-chip communication with wormhole switching," *Int. J. Emb. & Real-Time Comm. Syst.*, 2010.

[15] 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 *21st RTNS*, 2013.

[16] Z. Shi and A. Burns, "Priority assignment for real-time wormhole communication in on-chip networks," in *29th RTSS*, Dec 2008.

[17] M. Liu, M. Becker, M. Behnam, and T. Nolte, "Improved priority assignment for real-time communications in on-chip networks," in *23rd RTNS*, 2015.

[18] B. Nikolić and S. M. Petters, "Edf as an arbitration policy for wormhole-switched priority-preemptive nocs – myth or fact?" in *14th EMSOFT*, 2014.

[19] H. Kashif, S. Gholamian, and H. Patel, "Sla: A stage-level latency analysis for real-time communication in a pipelined resource model," *Trans. Computers*, vol. 99, April 2014.

[20] Z. Shi and A. Burns, "Schedulability analysis and task mapping for real-time on-chip communication," *Real-Time Syst. J.*, 2010.

[21] P. Mesidis and L. Indrusiak, "Genetic mapping of hard real-time applications onto noc-based mpsocs – a first approach," in *6th ReCoSoC*, 2011.

[22] A. Racu and L. Indrusiak, "Using genetic algorithms to map hard real-time on noc-based systems," in *7th ReCoSoC*, Jul 2012.

[23] M. Sayuti and L. Indrusiak, "Real-time low-power task mapping in networks-on-chip," in *IEEE Comp. Soc. Ann. Symp. VLSI*, 2013.

[24] L. S. Indrusiak, "End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration," *J. Syst. Arch.*, 2014.

[25] B. Nikolić, P. M. Yomsi, and S. M. Petters, "Worst-case communication delay analysis for many-cores using a limited migrative model," in *20th RTCSA*, 2014.

[26] A. Burns, J. Harbin, and L. Indrusiak, "A wormhole noc protocol for mixed criticality systems," in *35th RTSS*, 2014.

[27] L. S. Indrusiak, J. Harbin, and A. Burns, "Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip," in *27th ECRTS*, 2015.

[28] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1959.