

Using non-preemptive regions and path modification to improve schedulability of real-time traffic over priority-based NoCs

Meng Liu¹  · Matthias Becker¹ · Moris Behnam¹ · Thomas Nolte¹

Published online: 12 June 2017

© The Author(s) 2017. This article is an open access publication

Abstract Network-on-Chip (NoC) is a preferred communication medium for massively parallel platforms. Fixed-priority based scheduling using virtual-channels is one of the promising solutions to support real-time traffic in on-chip networks. Most of the existing works regarding priority-based NoCs use a flit-level preemptive scheduling. Under such a mechanism, preemptions can only happen between the transmissions of successive flits but not during the transmission of a single flit. In this paper, we present a modified framework where the non-preemptive region of each NoC packet increases from a single flit. Using the proposed approach, the response times of certain traffic flows can be reduced, which can thus improve the schedulability of the whole network. As a result, the utilization of NoCs can be improved by admitting more real-time traffic. Schedulability tests regarding the proposed framework are presented along with the proof of the correctness. Additionally, we also propose a path modification approach on top of the non-preemptive region based method to further improve schedulability. A number of experiments have been performed to evaluate the proposed solutions, where we can observe significant improvement on schedulability compared to the original flit-level preemptive NoCs.

This work is an extended version of the paper presented at RTCSA 2016 (Liu et al. 2016).

✉ Meng Liu
meng.liu@mdh.se

Matthias Becker
matthias.becker@mdh.se

Moris Behnam
moris.behnam@mdh.se

Thomas Nolte
thomas.nolte@mdh.se

¹ Mälardalen University, Västerås, Sweden

Keywords Network-on-Chip · Real-time traffic · Response time analysis · Routing

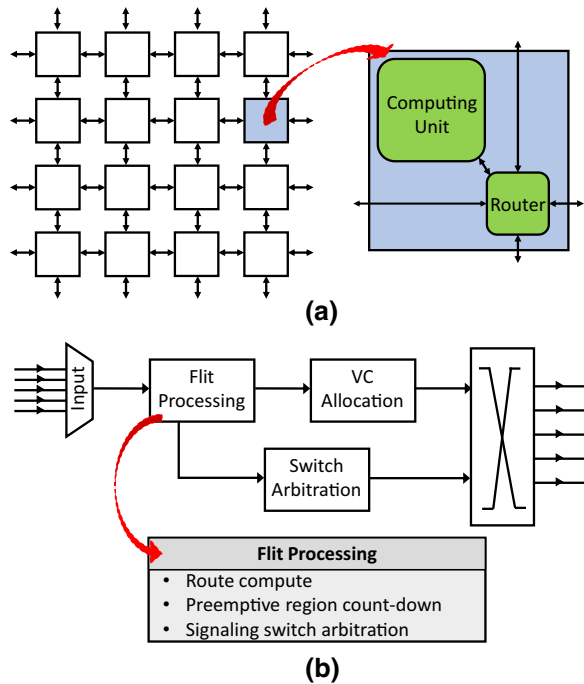
1 Introduction

Many-core processors are gaining a growing attention in industry because of their high computing capability along with limited hardware cost. Such a platform consists of a large number of components interconnected with each other. The data communication between components is typically achieved by a Network-on-Chip (NoC) (Benini and DeMicheli 2002). Wormhole-switching is a widely used technique in the existing NoC implementations (Ni and McKinley 1993). Under such a mechanism, once a router receives the header flit of a packet, the router can start to transmit without waiting for the arrival of the whole packet. Consequently, each router requires much smaller buffers compared to a store-and-forward communication network (e.g. Ethernet). A number of network topologies have been designed for NoCs, among which the 2-dimensional mesh-based architecture is the most commonly used. In such NoCs, adjacent nodes are typically connected by unidirectional channels through routers located on each node (e.g. Fig. 1a).

Different NoC designs have been proposed in the literature (e.g. Wentzlaff et al. 2007; Baron 2010; De Dinechin et al. 2014). In this paper, we focus on virtual-channel (VC) based NoCs (Dally 1992). Using the VC technique, each physical channel is shared by multiple VCs. A transmission control can be achieved at the output port of each router, so that preemptions between VCs can be supported. In most of related works using the same NoC architecture, preemptions can only happen between the transmission of different flits, but not during the transmission of a single flit. In Song et al. (1997), the authors have examined the applicability of VC based wormhole-switched NoCs for real-time traffic.

For real-time applications, the performance does not only rely on the functional correctness, but it also depends on the timeliness. In other words, a number of specific time requirements must be fulfilled (e.g. a packet needs to be transmitted within a given time duration called *deadline*), otherwise the system may get a degraded performance or it may even suffer catastrophic consequences. Therefore, while running real-time applications on NoC based many-core platforms, the time properties need to be considered carefully. Due to this reason, during the design phase of a real-time system, designers need to verify if all the time requirements can be satisfied. In order to provide such a guarantee, real-time NoCs may only admit a certain amount of traffic which is much lower than the capacity that the network can actually handle. In other words, the designed networks may not be sufficiently utilized, which may cause a waste of resource. Therefore, in this paper, we propose a new framework of VC based wormhole-switched NoCs in order to improve the schedulability of the whole network, so that more real-time traffic can be admitted. The proposed solution introduces a non-preemptive region to each NoC packet, so that the response time (also known as end-to-end latency or traversal time) of a packet can be potentially reduced without causing any deadline miss of other packets. To select proper sizes of non-preemptive regions, two heuristic approaches are proposed. Moreover, we also present a path modification approach on top of the above solution to further save flows from missing their

Fig. 1 **a** An example of a 2D-mesh based NoC. **b** Abstracts the router architecture in order to support limited-preemptive scheduling



deadlines when only applying non-preemptive regions is not sufficient. According to the evaluation results, the proposed method of using non-preemptive regions achieves higher schedulability ratios compared to the flit-level preemptive NoCs. The path modification approach can further provide more significant improvement on schedulability, however, it requires much more processing time especially when the network contains a large number of flows.

1.1 Contributions

This paper includes the following contributions:

- We present a new NoC framework where non-preemptive regions are introduced to NoC packets in order to improve the schedulability of real-time traffic.
- Two heuristic approaches are proposed for selecting proper sizes of non-preemptive regions.
- A sufficient schedulability test of the proposed framework is provided.
- A path modification method is presented to further improve schedulability.
- A number of experiments are implemented, including analysis based tests, simulation based tests and an industrial case study. The results clearly show the improvement of the proposed solutions regarding schedulability compared to the original flit-level preemptive NoCs.

1.2 Related work

For real-time applications, it is important that the time behaviors are predictable. In order to guarantee the predictability of NoCs, a number of research works have been proposed in the literature, such as the \mathcal{A} ethereal NoC (Goossens et al. 2005), the Time-Triggered Network-on-Chip (Paukovits and Kopetz 2008), the Back Suction flow-control scheme (Diemer and Ernst 2010), and fixed-priority based NoCs (Shi and Burns 2008, 2009b).

To verify if the given time requirements can be satisfied, suitable schedulability tests are necessary during the design phase of a real-time system. Methods to compute the end-to-end latency of a packet in round-robin arbitrated NoCs have been presented in Ferrandiz et al. (2009), Dasari et al. (2014). Targeting fixed-priority based NoCs, the authors in Shi and Burns (2008, 2009a, b) present a Response Time Analysis (RTA) based on the well-known analysis for task scheduling (Joseph and Pandya 1986). Kashif et al. (2015) have proposed a stage-level latency analysis which can provide tighter estimates of latencies compared to the analysis presented in Shi and Burns (2008). However, the stage-level analysis is based on a condition that, the buffer at each router is large enough such that the transmission of a packet cannot be delayed because the buffer at a downstream router is full. Unfortunately, due to the design principle of NoCs, the buffer size of each router is typically quite small (e.g. only holding several flits (Wentzlaff et al. 2007; Baron 2010) which does not support the above condition. When the buffer size is not sufficient, the analysis in Kashif et al. (2015) can produce optimistic results. Recently, the authors in Xiong et al. (2016) have shown that the analysis of Shi and Burns (2008) can be optimistic when a NoC buffer can hold more than a single flit, and they proposed a new method to resolve the optimistic problem. The network considered in this paper uses a fixed-priority based scheduling policy. Our proposed analysis is based on the work in Shi and Burns (2008, 2009a) with modifications to support the new features.

Task scheduling on a single-core processor has been well-studied in the past decades. Recently, limited preemptive scheduling receives a growing attention as a generalization of the existing fully-preemptive and non-preemptive scheduling, which can achieve better schedulability compared to the traditional fixed-priority based scheduling. The existing limited-preemptive scheduling uses three main approaches: (1) changing the priority of a task once the task is executed (e.g. Saksena and Wang 2000; Wang and Saksena 1999); (2) executing certain parts of a task in a non-preemptive manner (e.g. Bril et al. 2009; Bertogna et al. 2011a, b; Davis and Bertogna 2012); (3) a combination of the two former approaches (e.g. Bril et al. 2012). All these works target task scheduling on processors. However, to the best of our knowledge, such a technique has not been applied on data communication over NoCs. Therefore, in this paper, we aim to investigate the combination of wormhole-switched NoCs and the limited preemptive scheduling for real-time traffic. We introduce a non-preemptive region to each NoC packet, and we present algorithms to select suitable lengths of such regions.

1.3 Organization

The remainder of this paper is organized as follows. In Sect. 2, we present the network model considered in this paper. The transmission policies for packets with non-preemptive regions are described in Sect. 3. Section 4 recapitulates the existing RTA for fixed-priority based NoCs. Section 5 illustrates more details of the proposed non-preemptive region based framework where we present different algorithms to select the lengths of non-preemptive regions. In Sect. 6, we introduce the path modification method based on the above framework. The evaluation of the proposed approach is presented in Sect. 7. Finally, in Sect. 8 we draw conclusions along with some ideas about future work.

2 Network model

In this paper we focus on a wormhole switched NoC with a 2D-mesh based topology. An example is shown in Fig. 1a, where each node contains a computing unit along with a router. Each pair of adjacent nodes are connected by a full-duplex link with a fixed bandwidth.

The network contains a set \mathcal{F} of n periodic or sporadic real-time packet flows denoted as $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$. A flow refers to a series of packets that have the same characteristics. Each flow contains an infinite number of packets (also called *instances* hereinafter). A flow f_i can be characterized as $f_i = \{L_i, T_i, D_i, P_i, \mathcal{R}_i\}$. L_i represents the complete packet size of f_i including all the necessary segments (e.g. payload, header, tail-flit, etc.). T_i denotes the period if f_i is periodic, or the minimum inter-arrival time if f_i is sporadic. Each flow has an arbitrary relative deadline, which is denoted by D_i . A flow f_i is defined as *schedulable*, if its response time R_i is no larger than the deadline D_i (i.e. $R_i \leq D_i$). The flow set \mathcal{F} is defined as schedulable, if all the flows in \mathcal{F} can meet their deadlines ($R_i \leq D_i, \forall f_i \in \mathcal{F}$). P_i represents the unique¹ fixed-priority of f_i . Each flow f_i has a fixed transmission route/path (denoted as \mathcal{R}_i) which includes all the links from the source node (denoted as Sr_i) until the destination node (denoted as Ds_i). We assume that the buffer size of each router is one flit in order to simplify the current time analysis. An extension considering arbitrary buffer sizes will be conducted in the future work.

3 Transmission policy

In this section, we present the transmission policy used in the proposed framework where packets can have non-preemptive regions, along with some discussions regarding the hardware support.

The flit-level preemptive scheduling has been presented in many earlier works. Under such a mechanism, once a router receives a single flit of a packet, the router

¹ In this paper, we only assume distinct priorities. However, we need to clarify that using a priority sharing policy does not require any change in the proposed framework. Alternatively, the timing analysis and the selection of non-preemptive regions need to be modified.

can start to transmit the received flit without waiting for the arrival of the complete packet. The routing information is commonly stored in the header flit of each packet. When a router receives a header flit, the router needs to determine the following transmission route and assign the packet to a specific VC. The following flits can thus be transmitted directly using the assigned VC without requiring a complete processing. Since the transmission unit is flit, preemptions cannot occur during the transmission of a single flit, but can happen between the transmissions of two successive flits. Thus this type of scheduling is named flit-level preemptive scheduling.

Our proposed framework is based on the flit-level preemption with a modification of non-preemptive regions. This approach is inspired by the work presented in Bertogna et al. (2011a), where the authors show that by adding non-preemptive regions to tasks, the schedulability of the whole fixed-priority based task set can be effectively improved.

In a traditional flit-level preemptive NoC, each flit can be considered as a non-preemptive region, since its transmission cannot be preempted. In our framework, we group a number of successive flits into one larger non-preemptive region. To simplify the implementation as well as the analysis, the non-preemptive region is always placed at the end of each packet. The remaining flits, which are ahead of the non-preemptive region, will be transmitted as usual using the flit-level preemptive scheduling. These flits are thus called the preemptive region of a packet hereinafter. This approach is in between of the flit-level preemptive scheduling and the complete non-preemptive scheduling, thus it can be named as limited preemptive scheduling.

An abstract of the router architecture is presented in Fig. 1b, which is similar to the router proposed in the Intel SCC (Baron 2010). The main structure is the same as found in routers used in other priority based wormhole-switched NoCs. Additionally, the flit processing block of each router requires a modification, in order to support the limited preemptive scheduling.

The size of the non-preemptive region of each packet is computed offline (more details in Sect. 5) and stored in the header flit. When the header flit of a packet arrives at a router, the router will set a counter of the preemptive region. The counter is used to monitor the distance to the non-preemptive region of this packet. When the non-preemptive region has not been reached, the flits are transmitted using the flit-level preemptive scheduling where the utilized VC uses its initial priority level. Once the non-preemptive region is about to be transmitted on an output link, the corresponding VC will switch to the highest priority level such that no other packet can preempt at this router. Consequently, at each router, only one packet can be in the non-preemptive state. The event to promote the priority of a packet is implicitly propagated by the packet itself and hence affects each router independently. In other words, no synchronization is required between routers. When the last flit of the non-preemptive region has passed, the corresponding VC will switch back to its original priority level. Since only one VC at each router can have non-preemptive transmission at a time, we only need to reserve one additional priority level at each router which is higher than the initial priorities of all the VCs.

Note that when a packet is in the non-preemptive state at a certain router, the proposed method only guarantees that this packet cannot be preempted at this router and all upstream routers (i.e. the routers along the path towards the source node). However, this packet can suffer preemptions at downstream routers (i.e. the routers

along the path towards the destination node), because the preemptive region transmitted ahead can still be preempted which can further delay the transmission of the non-preemptive region. In other words, when the transmission of a packet reaches its non-preemptive region at a router, this packet is not completely non-preemptive. This is one of the main differences between non-preemptive regions in packets over NoCs and non-preemptive regions in tasks on single-core processors. When a task on a single-core processor reaches its non-preemptive region, it cannot be preempted by any other tasks. Therefore, the effect of utilizing non-preemptive regions on NoC packets may not be as significant as on tasks on single-core processors.

4 Recapitulate the RTA for flit-level preemptive NoCs

In Shi and Burns (2008), the authors have presented a RTA for NoCs with distinct priorities. This analysis is extended later in Shi and Burns (2009a) to support a priority sharing policy as well as arbitrary deadlines based on the results in Lehoczky (1990). However, the analysis in Shi and Burns (2009a) does not include the blocking delay caused by lower priority flows, and the included delays caused by the priority sharing mechanism is not necessary for this paper. Therefore, we first present a modified analysis based on the work presented in Shi and Burns (2009a).

Similar to the analysis presented in Shi and Burns (2009a), the maximum length of the i -level busy-period can be computed as

$$W_i = B_i + \left\lceil \frac{W_i}{T_i} \right\rceil \cdot C_i + I_i(W_i) \quad (1)$$

where B_i , C_i and I_i represent the blocking delay (caused by flows with lower priorities), the basic transmission time, and the interference (caused by flows with higher priorities) of f_i respectively.

The blocking delay B_i can be calculated as

$$B_i = \sum_{\forall \mathcal{L} \in \mathcal{R}_i} \max_{\substack{(\forall f_p \in S_i^B) \\ \wedge (\mathcal{L} \in \mathcal{R}_p)}} \delta_p \quad (2)$$

where S_i^B is a flow set including all the flows which can cause blocking to f_i , and δ_p represents the length of the blocking that f_p can cause to f_i at each hop. Under the flit-level preemptive mechanism, δ_p equates to the transmission time of a single flit over one link (denoted as τ).

The basic transmission time (i.e. without any blocking or interference) of f_i over \mathcal{R}_i (denoted by C_i) is computed as

$$C_i = \left(\left\lceil \frac{L_i}{\zeta} \right\rceil + \text{noh}(S_{r_i}, D_{s_i}) \right) \cdot \tau \quad (3)$$

where ζ denotes the size of a single flit, and $\text{noh}(S_{r_i}, D_{s_i})$ represents the *number of hops* along the route of f_i .

If a flow f_j ($P_j > P_i$) shares certain links with f_i , the transmission of f_j can increase the response time of f_i by causing interference. This type of interference is called *direct interference*. However, even if a flow f_x ($P_x > P_j > P_i$) does not share any link with f_i , f_x can still affect the response time of f_i . Such behavior can happen when f_x share links with f_j . By causing direct interference to f_j , f_x can change the minimum inter-arrival times between instances of f_j , which can further affect the response time of f_i . This type of interference that f_x causes to f_i is named *indirect interference*. In Shi and Burns (2008), the authors show that the effect of indirect interference can be bounded by adding an extra jitter (named *interference jitter*, denoted by J_j^I) to f_j during the analysis of f_i . Such a jitter of f_j can be upper-bounded by $R_j - C_j$.

The interference delay of f_i within a time duration of W_i can then be computed as

$$I_i(W_i) = \sum_{\forall f_j \in S_i^D} \left\lceil \frac{W_i + J_j^I}{T_j} \right\rceil \cdot C_j \tag{4}$$

where S_i^D represents the set of flows which can cause direct interference to f_i .

Equation 1 can be calculated using fixed-point iterations (Joseph and Pandya 1986). The calculation starts with $W_i(0) = B_i + C_i$, and terminates when the computed W_i converges (i.e. the minimum n is found that $W_i(n - 1) = W_i(n)$).

Once W_i is calculated, the number of instances within the largest i-level busy-period can be computed accordingly as

$$K_i = \left\lceil \frac{W_i}{T_i} \right\rceil \tag{5}$$

Assume that the first instance of f_i (i.e. denoted by $f_{i,1}$) is released at time 0, then the *finishing time* of any instance $f_{i,k}$ ($\forall k \in K_i$) within the i-level busy-period can be calculated as

$$F_{i,k} = B_i + kC_i + I_i(F_{i,k}) \tag{6}$$

The response time of each instance can then be computed as

$$R_{i,k} = F_{i,k} - (k - 1) \cdot T_i \tag{7}$$

Accordingly, the Worst-Case Response Time (WCRT) is calculated as

$$R_i = \max_{\forall k \in [1, K_i]} (R_{i,k}) \tag{8}$$

5 Scheduling NoC packets with non-preemptive regions

As presented in Sect. 3, in our proposed framework, a non-preemptive region is introduced to each NoC packet which is placed at the end of each packet. With an increased non-preemptive region, a flow can get less interference, but it can also cause

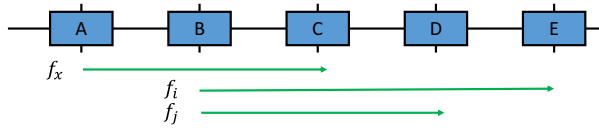


Fig. 2 An example of NoC flows with non-preemptive regions

more blocking to higher priority flows at the same time. Therefore, the sizes of non-preemptive regions need to be carefully selected.

We first revise the RTA presented in Sect. 4 to support non-preemptive regions in NoC flows. Then we propose a solution to compute the *blocking tolerance* of each flow. Finally, the size of the non-preemptive region of each flow can be selected based on the computed blocking tolerance.

5.1 Extended RTA of packets with non-preemptive regions

On a single core processor, once the execution of a non-preemptive region of a certain task starts, this execution cannot be preempted by any other task. However, on a wormhole-switched NoC, when a non-preemptive region of a certain flow starts its transmission, it can still be preempted. For example, assume that f_x has the lowest priority in the example shown in Fig. 2. Assume that a non-preemptive region of f_x starts its transmission on *Link(A, B)* at time t , and the head of the non-preemptive region arrives at node B at t_1 . Flow f_i which has higher priority than f_x arrives at node B at time t_2 which is slightly earlier than t_1 (i.e. $t < t_2 < t_1$). In this case, f_x has to wait until f_i releases *Link(B, C)*, even though f_i arrives at the route of f_x during the transmission of the non-preemptive region of f_x .

Thus, we point out a fact that *the non-preemptive region of a flow f_i is completely non-preemptive, when the head of the non-preemptive region starts its transmission on the output-link of N_i^{lastI} . N_i^{lastI} is the first node in \mathcal{R}_i , after which no more flows with higher priorities can encounter f_i for the first time*. For example, assume that f_i has the lowest priority in the example shown in Fig. 2. N_i^{lastI} will be node B, since f_x and f_j encounter f_i at node B for the first time. Once the non-preemptive region of f_i starts its transmission on *Link(B, C)*, it cannot be preempted any more.

Based on the above discussion, we divide the response time of the k th instance of f_i into two parts:

$$R_{i,k} = R_{i,k}^{pe} + R_{i,k}^{npe} \tag{9}$$

(1) the first part (denoted as $R_{i,k}^{pe}$) is the time duration since $f_{i,k}$ is released on the network until the non-preemptive region (i.e. the last segment of a packet) is about to be transmitted on N_i^{lastI} , during which the transmission of $f_{i,k}$ is still flit-level preemptive; (2) the second part (denoted as $R_{i,k}^{npe}$) begins when the non-preemptive region starts its transmission on N_i^{lastI} until $f_{i,k}$ completely arrives at Ds_i (i.e. the transmission of the whole packet is completed), during which the transmission of $f_{i,k}$ is completely non-preemptive. The constitution of $R_{i,k}$ is illustrated in Fig. 3.

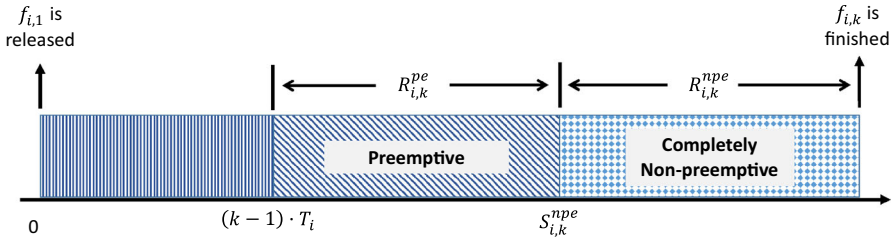


Fig. 3 The response time of $f_{i,k}$

For the flit-level preemptive part, the calculation is similar to Eqs. 1–8. We use $S_{i,k}^{npe}$ to represent the *starting time* of the non-preemptive region of a packet $f_{i,k}$ ($\forall k \in K_i$) (see Fig. 3). Assume that $f_{i,1}$ is released at time 0, all the preemptions arrived before $S_{i,k}^{npe}$ are possible to delay $f_{i,k}$. Thus, the finishing time of $f_{i,k}$ (according to Eq. 6) can be computed as:

$$F_{i,k} = B_i + kC_i + \sum_{\forall f_j \in S_i^D} \left\lceil \frac{S_{i,k}^{npe} + J_j^I}{T_j} \right\rceil \cdot C_j$$

As shown in Fig. 3, $S_{i,k}^{npe}$ can then be computed by:

$$\begin{aligned}
 S_{i,k}^{npe} &= F_{i,k} - R_i^{npe} = B_i + kC_i + I_i(S_{i,k}^{npe}) - R_i^{npe} \\
 I_i(S_{i,k}^{npe}) &= \sum_{\forall f_j \in S_i^D} \left\lceil \frac{S_{i,k}^{npe} + J_j^I}{T_j} \right\rceil \cdot C_j
 \end{aligned}
 \tag{10}$$

where K_i is computed using Eq. 5, and the calculation of R_i^{npe} will be presented in Sect. 5.3. The $R_{i,k}^{pe}$ of $f_{i,k}$ can then be computed by:

$$R_{i,k}^{pe} = S_{i,k}^{npe} - (k - 1) \cdot T_i \tag{11}$$

The response time of $f_{i,k}$ is therefore:

$$R_{i,k} = S_{i,k}^{npe} - (k - 1) \cdot T_i + R_i^{npe} \tag{12}$$

We can then obtain the WCRT of f_i using Eq. 8.

Based on the above analysis, we can prove that the response time of a NoC flow can be minimized by selecting its non-preemptive region as large as possible.

Theorem 1 *Decreasing the length of the non-preemptive region of a flow f_i in a fixed-priority based wormhole-switched NoC cannot decrease the response time of f_i while all the other parameters remain the same.*

Proof Assume that the original transmission time of the non-preemptive region of a packet $f_{i,k}$ is R_i^{npe} , and $f_{i,k}$ has the worst-case response time. We need to prove that when (as a result of decreasing the length of the non-preemptive region) we decrease R_i^{npe} to $R_i^{npe'} = R_i^{npe} - \Delta r$ ($0 < \Delta r < R_i^{npe}$), the new response time $R'_{i,k}$ becomes no smaller than the original one $R_{i,k}$ (i.e. $R'_{i,k} \geq R_{i,k}$). This can be proven using induction.

Equation 10 is solved using fixed-point iterations, which starts with $S_{i,k}^{npe}(0) = B_i + kC_i - R_i^{npe}$. Accordingly, we can also get

$$\begin{aligned} S_{i,k}^{npe'}(0) &= B_i + kC_i - (R_i^{npe} - \Delta r) \\ &= S_{i,k}^{npe}(0) + \Delta r \end{aligned}$$

Then in the second iteration, we have

$$\begin{aligned} S_{i,k}^{npe}(1) &= B_i + kC_i - R_i^{npe} + I_i(S_{i,k}^{npe}(0)) \\ S_{i,k}^{npe'}(1) &= B_i + kC_i - R_i^{npe'} + I_i(S_{i,k}^{npe'}(0)) \end{aligned}$$

Since $I_i(S_{i,k}^{npe})$ is a non-decreasing function, we can obtain that

$$\begin{aligned} I_i(S_{i,k}^{npe'}(0)) &\geq I_i(S_{i,k}^{npe}(0)) \\ \Rightarrow B_i + kC_i - R_i^{npe'} + I_i(S_{i,k}^{npe'}(0)) & \\ \geq B_i + kC_i - R_i^{npe} + \Delta r + I_i(S_{i,k}^{npe}(0)) & \\ \Rightarrow S_{i,k}^{npe'}(1) \geq S_{i,k}^{npe}(1) + \Delta r & \end{aligned}$$

Now assume that $S_{i,k}^{npe'}(q) \geq S_{i,k}^{npe}(q) + \Delta r$, we need to prove that $S_{i,k}^{npe'}(q + 1) \geq S_{i,k}^{npe}(q + 1) + \Delta r$. Due to the non-decreasing property of $I_i(S_{i,k}^{npe})$, we can get

$$\begin{aligned} I_i(S_{i,k}^{npe'}(q)) &\geq I_i(S_{i,k}^{npe}(q)) \\ \Rightarrow B_i + kC_i - R_i^{npe'} + I_i(S_{i,k}^{npe'}(q)) & \\ \geq B_i + kC_i - R_i^{npe} + \Delta r + I_i(S_{i,k}^{npe}(q)) & \\ \Rightarrow S_{i,k}^{npe'}(q + 1) \geq S_{i,k}^{npe}(q + 1) + \Delta r & \end{aligned}$$

Therefore, given $R_i^{npe'} < R_i^{npe}$, we have $S_{i,k}^{npe'} \geq S_{i,k}^{npe} + \Delta r$. According to Eq. 12, we can obtain that

$$\begin{aligned} S_{i,k}^{npe'} - (k - 1) \cdot T_i + R_i^{npe'} &\geq S_{i,k}^{npe} - (k - 1) \cdot T_i + R_i^{npe'} + \Delta r \\ \Rightarrow R'_{i,k} &\geq R_{i,k} \end{aligned}$$

This completes the proof. □

According to Theorem 1, when we increase the size of the non-preemptive region of a certain flow f_i , f_i can have a shorter response time. However, the flows with priorities higher than P_i may get larger response times, because they may suffer more blocking from f_i . Therefore, the non-preemptive region of each flow cannot be infinitely large (up to the size of the whole packet), and it should be carefully investigated by balancing the effects on all related flows.

5.2 Computing the blocking tolerance

In this section, we present the computation of the blocking tolerance of each flow. The blocking tolerance of f_i (i.e. denoted by β_i) is the maximum blocking delay that f_i can tolerate without causing a deadline miss. In other words, the flows in S_i^B should not cause blocking to f_i for more than β_i .

First, we reformat the response time analysis for NoC flows in order to simplify the following presentation. In the above RTA, the purpose of using the fixed-point based calculations is to iteratively compute a convergency of the i-level busy-period. If the computed i-level busy-period at a certain iteration can reach the release time of a new packet, the analysis will continue to the next iteration where the i-level busy-period will become larger. On the other hand, if the same computed i-level busy-period is obtained from two successive iterations (i.e. the computed i-level busy-period cannot reach the release time of any packet with higher priorities), the analysis can be terminated. Therefore, the analysis is actually regarding the length of the i-level busy-period and the release times of flows with higher priorities. We define $\Pi_{i,k}$ as the set of releases times of flows in S_i^D within the time duration since a packet $f_{i,k}$ is released with its critical instant until the deadline of $f_{i,k}$ is reached. Assume that the i-level busy-period starts at time 0, then $\Pi_{i,k}$ can be defined as

$$\begin{aligned} \Pi_{i,k} = & [(k - 1)T_i, (k - 1)T_i + D_i - R_i^{npe}] \\ & \cap \left\{ \max(0, (p - 1)T_j - J_j^I), \forall p \in \mathbb{N}, \forall f_j \in S_i^D \right\} \\ & \cup \left\{ (k - 1)T_i + D_i - R_i^{npe} \right\} \end{aligned}$$

Note that the scenario of $f_{i,k}$ just reaching its deadline also needs to be taken into account in $\Pi_{i,k}$.

Then the RTA for NoC flows can be reformatted as follows.

Theorem 2 *A NoC flow f_i is schedulable if $\forall k \in [1, K_i], \exists t \in \Pi_{i,k}$, such that*

$$B_i + kC_i - R_i^{npe} + I_i(t) \leq t$$

Proof The i-level busy-period consists of the blocking delay (B_i), the transmission of f_i itself (kC_i), and the interference from flows in S_i^D (I_i). As discussed in Sect. 5.1, once the non-preemptive region of a packet $f_{i,k}$ starts its transmission after node N_i^{lastI} , later-arrived packets with higher priorities cannot preempt $f_{i,k}$ anymore. Therefore, given a time duration $[0, t]$, only the workload of $B_i + kC_i - R_i^{npe} + I_i(t)$ actually affects if new arrivals of interference need to be considered.

When the workload of $B_i + kC_i - R_i^{npe} + I_i(t)$ exceeds the length of t , the non-preemptive region of $f_{i,k}$ cannot start its transmission at N_i^{lastI} . In this case, the new arrivals which are released at time instant t can still preempt $f_{i,k}$, which will contribute to the growth of the i -level busy-period. Consequently, the computation will not terminate at time t .

When $B_i + kC_i - R_i^{npe} + I_i(t) \leq t$, the workload of $B_i + kC_i - R_i^{npe} + I_i(t)$ can be finished before time instant t . In this case, the non-preemptive region of $f_{i,k}$ can start its transmission before or at t . Therefore, the packets arrived at or later than t cannot affect the i -level busy-period anymore. According to the definition of $\Pi_{i,k}$, the maximum t is $(k - 1)T_i + D_i - R_i^{npe}$. Thus, $B_i + kC_i - R_i^{npe} + I_i(t) \leq t$ implies that the packet $f_{i,k}$ which is released at time $(k - 1)T_i$ must finish its transmission no later than its deadline. □

According to Theorem 2, in order to guarantee that any packet $f_{i,k}$ can meet its deadline, we should have

$$\begin{aligned} \exists t \in \Pi_{i,k} : B_i + kC_i - R_i^{npe} + I_i(t) &\leq t \\ \Rightarrow B_i &\leq t - kC_i + R_i^{npe} - I_i(t) \end{aligned}$$

Since any t , that can be found satisfying the above condition, can make $f_{i,k}$ meet its deadline, we can observe an upper-bound of B_i as

$$\beta_{i,k} = \max_{t \in \Pi_{i,k}} (t - kC_i + R_i^{npe} - I_i(t)) \tag{13}$$

An upper-bound of B_i , which can guarantee that all the instances of f_i can meet deadlines, can then be computed as

$$\beta_i = \min_{k \in [1, K_i]} \beta_{i,k} \tag{14}$$

While calculating K_i using Eq. 5, we need to know the blocking delay (see Eq. 1). As a result, before computing β_i using Eq. 14, we need to know the value of K_i which depends on an unknown blocking delay. In order to solve such circular-dependency problem, we use a similar approach as presented in Bertogna et al. (2011a). The solution is based on the fact that

$$\begin{aligned} \beta_i &= \min_{\forall k \in [1, K_i]} \beta_{i,k} \\ \Rightarrow \beta_i &\leq \beta_{i,k}, \forall k \in [1, K_i] \end{aligned}$$

Since Eq. 1 is a non-decreasing function regarding the blocking delay, using β_i will not compute a larger i -level busy-period than using any other $\beta_{i,k}$. Therefore, we use $\beta_{i,1}$ to compute an approximate \hat{K}_i ($\hat{K}_i \geq K_i$).

The main procedure of computing β_i is presented in Algorithm 1. As discussed earlier, we first compute $\beta_{i,1}$ (line 2) which is used for the calculation of the approximate

\hat{K}_i . Our approach is proposed on top of the existing flit-level preemptive mechanism. Therefore, each flow has a certain amount of non-avoidable blocking delay due to basic flit-level preemptions. The set of links, where f_i may experience blocking delays, are denoted as φ_i , which is defined as

$$\varphi_i \stackrel{def}{=} \bigcup_{\forall f_j \in S_i^B} (\mathcal{R}_i \cap \mathcal{R}_j)$$

Such a basic blocking delay can be bounded by $nr(\varphi_i) \cdot \tau$, where $nr(\varphi_i)$ denotes the number of links in φ_i . If the computed $\beta_{i,1}$ is smaller than $nr(\varphi_i) \cdot \tau$, which means that $f_{i,1}$ may miss its deadline even only experiencing basic blocking, the algorithm can thus be terminated. If a valid $\beta_{i,1}$ is found, we can compute an approximate i-level busy-period using Eq. 1, as well as the \hat{K}_i using Eq. 5 (line 7). The blocking tolerance of each instance within the i-level busy-period can then be calculated using Eq. 13 (line 9-14). Similar to the discussion of $\beta_{i,1}$, when any $\beta_{i,k}$ is smaller than $nr(\varphi_i) \cdot \tau$, the algorithm can be terminated. Finally, a suitable β_i can be selected by Eq. 14 (line 15).

Algorithm 1 Compute β_i

```

1: //compute  $\beta_{i,1}$ 
2:  $\beta_{i,1} \leftarrow$  using Eq. 13
3: if  $\beta_{i,1} < nr(\varphi_i) \cdot \tau$  then
4:   return 'Unschedulable'
5: end if
6: //compute  $\hat{K}_i$ 
7:  $\hat{K}_i \leftarrow$  using Eq. 5 and 1, where  $B_i = \beta_{i,1}$  in Eq. 1
8: //compute  $\beta_i$ 
9: for all  $k$  in  $[2, \hat{K}_i]$  do
10:   $\beta_{i,k} \leftarrow$  using Eq. 13
11:  if  $\beta_{i,k} < nr(\varphi_i) \cdot \tau$  then
12:    return 'Unschedulable'
13:  end if
14: end for
15:  $\beta_i \leftarrow$  using Eq. 14
16: return  $\beta_i$ 

```

5.3 Selecting the lengths of non-preemptive regions

In this section, we present how to select the length of the non-preemptive region of each NoC flow. We use L_i^{npe} to represent the length of the non-preemptive region of f_i . The blocking delay, that f_i can cause to any flows with higher priorities over one physical link, can then be computed as

$$c_i^{npe} = \frac{L_i^{npe}}{\varsigma} \cdot \tau \tag{15}$$

where ς represents the size of a single flit, and τ denotes the transmission time of a single flit over one link. Once c_i^{npe} is known, L_i^{npe} can also be computed based on

Eq. 15. Accordingly, R_i^{npe} can then be calculated as

$$R_i^{npe} = c_i^{npe} + \left(noh(N_i^{lastI}, Ds_i) - 1 \right) \cdot \tau \tag{16}$$

Therefore, the aim is to find a suitable c_i^{npe} for each flow f_i .

As proved in Theorem 1, increasing the length of the non-preemptive region of a flow can potentially decrease its response time. However, as the non-preemptive region of a flow increases, the related flows with higher priorities may have longer response times due to the increase of blocking delays. In Sect. 5.2, we have presented how to calculate the blocking tolerance of each flow. As long as the blocking delay of each flow does not exceed its blocking tolerance, this flow is still guaranteed to be schedulable. Therefore, the selection of the non-preemptive region of a flow f_i is based on the rule that *the blocking delay, that f_i can cause to any related flow f_j with a higher priority (i.e. $\forall f_j \in S_i^D$), should not exceed the blocking tolerance of f_j (i.e. β_j)*.

According to Eq. 2, we know that each NoC flow may experience blocking over multiple links. Therefore, in order to guarantee the schedulability of a flow f_i , the summation of all the blocking that f_i may suffer along its route should be upper-bounded by its blocking tolerance (i.e. $B_i \leq \beta_i$). In other words, the blocking tolerance of f_i needs to be distributed to all the links where blocking may occur. Based on different distributions of the blocking tolerance of a flow f_i , the selection of the non-preemptive regions of the flows in S_i^B will differ as well. In this paper, we present two different approaches to distribute the blocking tolerance of each flow.

5.3.1 Even distribution of blocking tolerance

In the first approach (named Even Distribution of Blocking Tolerance, EDBT), we evenly distribute the blocking tolerance of a flow f_i to all the links where f_i may get blocking (i.e. φ_i). Accordingly, the blocking tolerance assigned to each link in φ_i is computed as

$$\beta_i^u = \frac{\beta_i}{max(1, nr(\varphi_i))} \tag{17}$$

where $nr(\varphi_i)$ denotes the number of links in φ_i . When $nr(\varphi_i) = 0$, f_i will not experience any blocking. Therefore, we only need to consider the situations where $nr(\varphi_i)$ is at least 1.

At each link, the transmission time of a non-preemptive region of f_i should not cause blocking to any flow f_j ($\forall f_j \in S_i^D$) more than β_j^u (i.e. $c_i^{npe} \leq \beta_j^u$). In the best case, flow f_i can be completely non-preemptive. Therefore, the transmission time of the non-preemptive region of f_i over one link can be calculated as

$$c_i^{npe} = \min \left(c_i, \min_{\forall f_j \in S_i^D} (\beta_j^u) \right) \tag{18}$$

where c_i represents the maximum transmission time of a whole packet of f_i over one link which can be computed as $\frac{L_i}{\zeta} \cdot \tau$.

Algorithm 2 Schedulability Test with c_i^{npe} selected using EDBT

```

1: for all  $f_i \in \mathcal{F}$  in a descending order of priorities do
2:    $c_i^{npe} \leftarrow$  using Eq. 18
3:    $R_i^{npe} \leftarrow$  using Eq. 16
4:    $\beta_i \leftarrow$  using Algorithm 1
5:   if  $\beta_i < nr(\varphi_i) \cdot \tau$  /* Algorithm 1 returns 'Unschedulable' */ then
6:     return 'Unschedulable'
7:   end if
8: end for
9: return 'Schedulable'

```

The selection procedure is presented in Algorithm 2. Since Eq. 18 requires the blocking tolerance of flows with higher priorities, the algorithm starts from the flow with the highest priority (line 1). If the algorithm terminates without returning 'Unschedulable', the flow set is guaranteed to be schedulable with the selected non-preemptive regions.

5.3.2 Higher-priorities-favored distribution of blocking tolerance

In the second solution, we try to take the actual information of lower priority flows into account while distributing the blocking tolerance of each flow. This distribution algorithm (Algorithm 3) tries to provide more satisfaction to flows with higher priorities, thus it is named Higher-Priorities-favored Distribution of Blocking Tolerance (HPDBT).

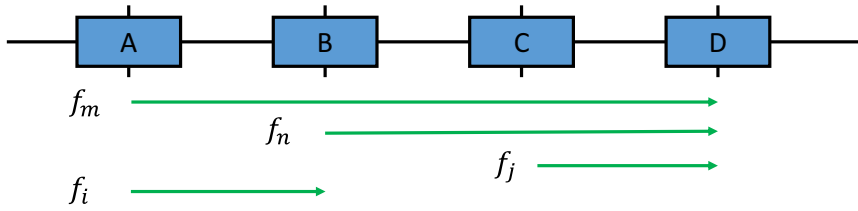
Algorithm 3 Schedulability Test with c_i^{npe} selected using HPDBT

```

1: for all  $f_i \in \mathcal{F}$  in a descending order of priorities do
2:    $c_i^{npe} \leftarrow$  using Eq. 19
3:   for all  $f_j \in S_i^D$  do
4:     for all  $l \in (\mathcal{R}_i \cap \mathcal{R}_j)$  do
5:        $\Delta\beta_{j,l} = \max(0, c_i^{npe} - \beta_{j,l})$ 
6:        $\beta_{j,l} = \max(\beta_{j,l}, c_i^{npe})$ 
7:        $\beta_j^- = \beta_j^- - \Delta\beta_{j,l}$ 
8:     end for
9:   end for
10:   $R_i^{npe} \leftarrow$  using Eq. 16
11:   $\beta_i \leftarrow$  using Algorithm 1
12:   $\beta_{i,l} = 0, \forall l \in \varphi_i$ 
13:   $\beta_i^- = \beta_i$ 
14:  if  $\beta_i < nr(\varphi_i) \cdot \tau$  /* Algorithm 1 returns 'Unschedulable' */ then
15:    return 'Unschedulable'
16:  end if
17: end for
18: return 'Schedulable'

```

As discussed earlier, the selection of the non-preemptive region of a flow depends on the blocking tolerance of flows with higher priorities as well. Therefore, while



- Assume:
- $\beta_m^- = 10$
 - $P_m > P_n > P_j > P_i$
 - After selecting the non-preemptive region of f_n , $\beta_{m, Link(B,C)} = 2, \beta_{m, Link(C,D)} = 2, \beta_m^- = 6$

Fig. 4 An example of applying HPDBT

calculating the lengths of non-preemptive regions, we start from the flow with the highest priority. In this approach, we separate the capacity of the blocking tolerance of f_i into two parts: the assigned blocking tolerance on each link l (denoted as $\beta_{i,l}, \forall l \in \varphi_i$), and the remaining blocking tolerance (denoted as β_i^-). Initially, $\beta_{i,l} = 0$ and $\beta_i^- = \beta_i$ (line 6–7).

We know that each packet can experience a blocking at most once on each link. In other words, the blocking tolerance of f_j assigned to a certain link l is shared by all the flows in S_j^B that also pass link l . Therefore, a flow $f_i (f_i \in S_j^B)$, which shares link l with f_j , can reuse the existing blocking tolerance of f_j which has already been assigned to l . If f_i requires more blocking tolerance from f_j in order to get the maximum non-preemptive region, the algorithm will try to assign an incremental blocking tolerance to the current one. Otherwise, the algorithm keeps the current blocking tolerance assigned on this link. The transmission time of the non-preemptive region of f_i over one link can thus be calculated as

$$c_i^{npe} = \min \left(c_i, \min_{\forall f_j \in S_i^D} \left(\min_{\forall l \in (\mathcal{R}_i \cap \mathcal{R}_j)} \left(\frac{\beta_j^-}{nr(\mathcal{R}_i \cap \mathcal{R}_j)} + \beta_{j,l} \right) \right) \right) \tag{19}$$

Once the non-preemptive region of f_i is selected, the corresponding blocking tolerance capacities of f_j need to be updated (line 9–15). An example showing how blocking tolerance is distributed under HPDBT is presented in Example 1.

Example 1 As shown in Fig. 4, assume that there are four flows in the network. We show how β_m is distributed under HPDBT. After the non-preemptive region selection for f_n , we get $\beta_{m, Link(B,C)} = 2, \beta_{m, Link(C,D)} = 2$ and $\beta_m^- = 6$. Now we consider the following two cases.

Case 1 Assume that c_j equates to 1.

In this case, the current distribution of β_m does not require any change, because the current $\beta_{m, Link(C,D)}$ is large enough for f_j to get its maximum c_j^{npe} . Consequently, f_i can utilize the blocking tolerance of f_m up to 6 (i.e. $\beta_{m, Link(A,B)}$ can be up to 6).

Case 2 Assume that c_j equates to 5.

In this case, the current $\beta_{m,Link(C,D)}$ is not sufficient for f_j to get its maximum c_j^{npe} . Thus, $\beta_{m,Link(C,D)}$ is increased to 5, and β_m^- is reduced to 3. As a result, f_i can only utilize the blocking tolerance of f_m up to 3.

The above selection procedure is repeated until all the flows have been processed. If Algorithm 3 terminates without returning 'Unschedulable', the whole flow set is schedulable with the selected non-preemptive regions.

As discussed earlier, if the blocking delay of a flow does not exceed its valid blocking tolerance, this flow is still guaranteed to be schedulable. On the other hand, according to Theorem 1, increasing the non-preemptive region of a flow cannot increase its response time. Therefore, if a flow is schedulable in the original flit-level preemptive NoC, it will still meet its deadline with the non-preemptive regions selected using the above proposed approaches. If a flow misses its deadline in the original flit-level preemptive NoC, it is potentially schedulable with increased non-preemptive regions selected using our approaches. In other words, our proposed solutions can only achieve a better or the same schedulability ratio compared to the original flit-level preemptive NoC, but cannot make it worse.

6 Path modification scheme

In Sect. 5, we have presented how to introduce non-preemptive regions to packets in priority-based NoCs aiming to improve the schedulability of the whole flow set. As explained earlier, the improvement highly depends on the *laxity*² of flows with higher priorities. The larger laxity a flow has, the more blocking this flow can tolerate (i.e. having a greater blocking tolerance). On the other hand, a flow with a smaller laxity can tolerate less blocking. Consequently, for some cases, utilizing non-preemptive regions cannot save flows from missing their deadlines because of little laxity of the flows with higher priorities. Therefore, targeting the flows still missing their deadlines after introducing non-preemptive regions, we need other approaches to further improve their schedulability.

The XY-routing policy is one of the most popular routing algorithms for NoCs with a 2D-mesh based topology (Cota et al. 2011). Since it is deadlock-free and easy to implement, it has been used by many processor vendors (e.g. Wentzlaff et al. 2007; Baron 2010). However, as discussed in Nikolic et al. (2016), XY-routing has the disadvantage that it does not flexibly distribute transmission workload in the network, which can result in inefficient utilization of bandwidth. An example is depicted in Fig. 5 to show such a disadvantage. As shown in Fig. 5a, under the XY-routing policy, all the three flows can cause contention to each other due to sharing the same physical link. However, if we change the routes of these flows according to Fig. 5b, the contention is reduced significantly. Thus, modifying the routes of flows in a NoC is a potential solution to save flows from missing their deadlines when only applying non-preemptive regions is not sufficient. In this section, we present a Path/route Modification (PM) approach to further improve the schedulability of NoC flows with non-preemptive

² The laxity of a flow f_i is the time distance from its finishing time until its deadline (i.e. $D_i - R_i$).

Fig. 5 An example showing the disadvantage of XY-routing

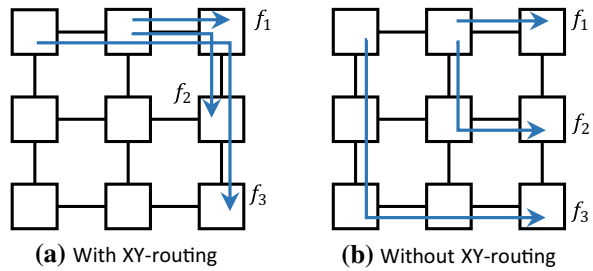
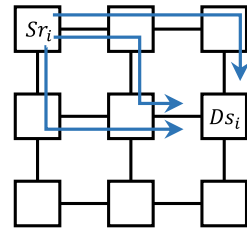


Fig. 6 An example showing route candidates



regions. Note, that the proposed path modification mechanism is independent of the non-preemptive region selection algorithms. To simplify the presentation, we use HPDBT as an example to present the path modification approach.

The path modification method is illustrated in Algorithm 4. We use XY-routing as the initial routing policy for the whole flow set (line 1). We then apply Algorithm 3 (or Algorithm 2) to select proper non-preemptive region for each flow (lines 3–7). If the algorithm gives a positive result showing that all the flows meet their deadlines, the current design can be directly approved without applying path modification (line 8 and 34). On the other hand, if the algorithm returns a negative result showing that a certain flow f_i is unschedulable (line 16), we start the path modification process to save f_i from missing its deadline.

First, we need to identify a number of route candidates for each flow. Apparently, if all the possible routes of f_i are considered as its route candidates, the search space can become huge as the number of routers increases. Therefore, in this work, the route candidate identification process follows a *minimal path* policy (Nikolic et al. 2016). Given the coordinates of the source and destination nodes of f_i (denoted as $[x_i^{Sr}, y_i^{Sr}]$ and $[x_i^{Ds}, y_i^{Ds}]$), the minimum number of hops involved in the route \mathcal{R}_i is $|x_i^{Sr} - x_i^{Ds}| + |y_i^{Sr} - y_i^{Ds}|$. Under the minimal path policy, the number of hops included in each route candidate equates to the minimum number of hops computed above. In Nikolic et al. (2016), the authors have discussed the benefits of the minimal path policy, including achieving constant isolation latencies, reducing solution space, simplifying implementation, and avoiding deadlocks. To achieve the minimal path policy, we need to follow the principle that a flow can only propagate towards its destination node. For example, if the destination node is located on the south-east of the source node, the transmission of this flow should never go towards the north or the west. An example is presented in Fig. 6 showing the route candidates of a flow

Algorithm 4 Schedulability Test with c_i^{npe} selected using HPDBT and path modification

```

1: Assign routes of all the flows in  $\mathcal{F}$  using XY-routing
2: for all  $f_i \in \mathcal{F}$  in a descending order of priorities do
3:    $c_i^{npe} \leftarrow$  using Eq. 19
4:    $R_i^{npe} \leftarrow$  using Eq. 16
5:    $\beta_i \leftarrow$  using Algorithm 1
6:    $\beta_{i,l} = 0, \forall l \in \varphi_i$ 
7:    $\beta_i^- = \beta_i$ 
8:   if  $\beta_i \geq nr(\varphi_i) \cdot \tau$  /* Algorithm 1 returns 'Schedulable' */ then
9:     for all  $f_j \in S_i^D$  do
10:      for all  $l \in (\mathcal{R}_i \cap \mathcal{R}_j)$  do
11:         $\Delta\beta_{j,l} = \max(0, c_i^{npe} - \beta_{j,l})$ 
12:         $\beta_{j,l} = \max(\beta_{j,l}, c_i^{npe})$ 
13:         $\beta_j^- = \beta_j^- - \Delta\beta_{j,l}$ 
14:      end for
15:    end for
16:  else
17:    /* Algorithm 1 returns 'Unschedulable', and path modification is required */
18:     $foundTolerance \leftarrow False$ 
19:    while  $(\beta_i < nr(\varphi_i) \cdot \tau) \wedge (RCL_i \neq \emptyset)$  do
20:      Get the route candidate with the best heuristic from  $RCL_i$ 
21:      Remove the selected candidate from  $RCL_i$ 
22:      Repeat line 3 to 7
23:      if  $\beta_i \geq nr(\varphi_i) \cdot \tau$  /* Algorithm 1 returns 'Schedulable' */ then
24:         $foundTolerance \leftarrow True$ 
25:        Repeat line 9 to 15
26:        Break
27:      end if
28:    end while
29:    if  $foundTolerance = False$  then
30:      return 'Unschedulable'
31:    end if
32:  end if
33: end for
34: return 'Schedulable'

```

with given source and destination nodes. We use RCL_i to represent the list of route candidates for f_i .

After identifying the route candidate list for each flow, we need to select one of the candidates for each path modification process. The authors in Nikolic et al. (2016) have shown that performing an exhaustive search to find the optimal route candidate is impractical even though the solution space has been reduced by the minimal path policy. Therefore, in this paper, we propose a heuristic based approach to select route candidates.

The utilized heuristic is to choose the route candidate which has the lowest number of flows from S_i^D passing through. Such a heuristic aims to find a route where f_i can experience less interference such that f_i has a higher probability to meet its deadline and to provide relatively large blocking tolerance for flows with lower priorities. Algorithm 5 demonstrates the heuristic based selection of a route candidate for f_i .

Algorithm 5 Heuristic based selection of route candidate for f_i

```

1: Input:  $S_i^D, RCL_i$ 
2:  $minIntFls \leftarrow sizeof(S_i^D)$ 
3:  $candidate \leftarrow null$ 
4: for all  $rc \in RCL_i$  do
5:    $nrIntFls \leftarrow 0$ 
6:   for all  $f_j \in S_i^D$  do
7:     if  $(\exists l_n \in rc) \wedge (l_n \in \mathcal{R}_j)$  then
8:        $nrIntFls ++$ 
9:     end if
10:  end for
11:  if  $nrIntFls \leq minIntFls$  then
12:     $minIntFls \leftarrow nrIntFls$ 
13:     $candidate \leftarrow rc$ 
14:  end if
15: end for
16: return  $candidate$ 

```

Once a route candidate of f_i is chosen, we repeat the selection process of the non-preemptive region for f_i (Algorithm 4, line 22). If f_i is still unschedulable with the current route candidate, the algorithm continues to check other candidates in RCL_i . The processing on f_i terminates when (1) f_i becomes schedulable, in which case the algorithm starts to analyze the next flow in \mathcal{F} (Algorithm 4, lines 23–26); or (2) there are not candidates remained in RCL_i , in which case the algorithm aborts returning a result of 'Unschedulable' (Algorithm 4, lines 29 and 30).

Similar to Algorithms 2 and 3, Algorithm 4 processes flows in a descending order of their priorities. This is used to guarantee that once a valid non-preemptive region of a flow f_i is selected, the later processing of other flows (including both non-preemptive region selection and path modification) cannot affect the schedulability of f_i . Thus, there is no need to recheck the schedulability of the flows which have been already processed. Moreover, if a flow is schedulable with its initial route, no path modification will be applied on it. If a flow is unschedulable with its initial route, the path modification will be performed which can potentially make this flow become schedulable. Therefore, we can conclude that using path modification can achieve either a higher or an equal schedulability ratio compared to a solution without path modification.

7 Evaluation

We have generated a number of experiments to evaluate the performance of the proposed approaches. The evaluation is implemented using Python 2.7. The experiments are performed on a computer using Window 7 and equipped with Intel i5-5200U@2.2 GHz CPU and 16 GB RAM.

7.1 Evaluation of EDBT and HPDBT

First, we present the evaluation outcomes of EDBT and HPDBT. The evaluation consists of analysis based tests, simulation based tests, along with an industrial case study based on an automotive application.

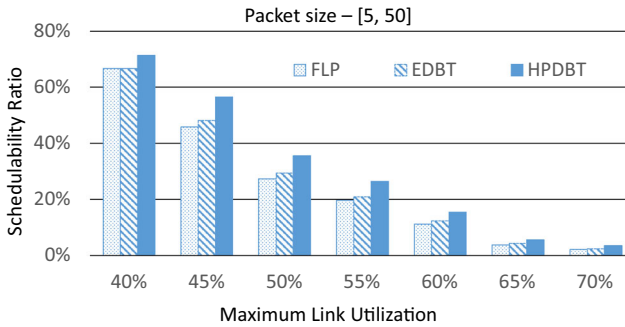


Fig. 7 Results regarding the maximum link utilization with packet size selected from [5, 50]

7.1.1 Analysis based evaluation

In this set of evaluations, we compare our proposed approaches (i.e. EDBT and HPDBT) with the original flit-level preemptive NoC (denoted by FLP) based on offline analyses.

The network considered in the evaluation uses a 8×8 2-D meshed topology, and it contains 100 flows. The source and the destination of each flow is randomly selected. All the flows are routed using the XY-routing algorithm which is widely supported in most of the existing NoC implementations. The size of each flow is randomly³ generated from the range of [5, 1000] flits. The utilization of each flow is randomly selected from [0.03, 10%]. The priorities of flows are assigned using the Rate Monotonic (RM) algorithm (Liu and Layland 1973). We generate two groups of experiments in order to investigate the effects of different system parameters.

In the first set of experiments, we show how network utilization and flow size affect the performance of the proposed approaches. The results are presented in Figs. 7, 8, 9. As shown in Fig. 7, as the maximum link utilization (i.e. link utilization denotes the total utilization of all the flows passing a certain link) goes up, the schedulability ratios of all the three approaches decrease. EDBT is slightly better than FLP, while HPDBT obviously dominates the other approaches. When the utilization is between 0.4 and 0.55, EDBT can improve the schedulability by around 3% compared to FLP, and HPDBT can improve the schedulability by more than 10%. Figures 8 and 9 show the results of experiments where we increase the sizes of flows. Similar to the observation from Fig. 7, the schedulability ratios decrease as the maximum link utilization goes up. HPDBT is always better than EDBT and FLP, while EDBT is slightly better than FLP.

In the second set of experiments, we investigate the effect of the path length of each flow. In this set of experiments, we increase the selection range of the utilization of each flow to [1, 10%], in order to show the results more clearly. As presented in Fig. 10, with the same setting of packet sizes and periods, the schedulability ratios decrease as the path lengths of flows increase. This is because when the path length of

³ In our experiments, all the randomly selected parameters are following a uniform distribution within the given ranges.

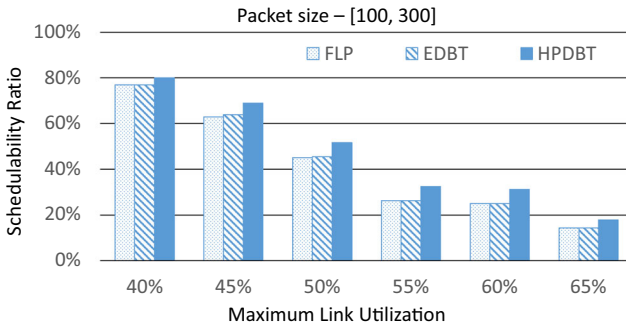


Fig. 8 Results regarding the maximum link utilization with packet size selected from [100, 300]

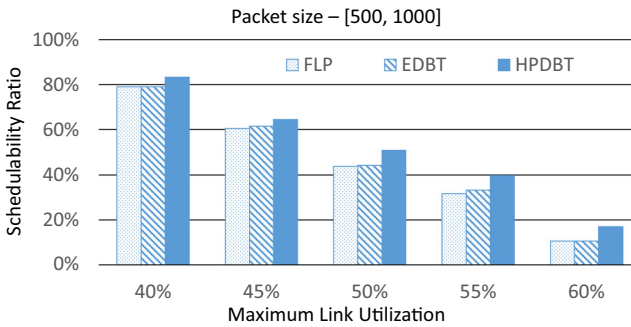


Fig. 9 Results regarding the maximum link utilization with packet size selected from [500, 1000]

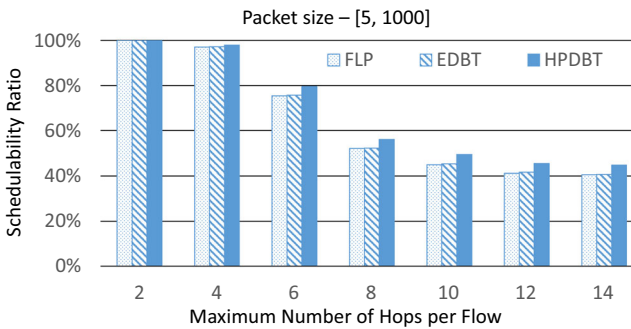


Fig. 10 Results regarding the maximum number of hops per flow with packet size selected from [5, 1000]

a flow f_i increases, f_i can experience more blocking (i.e. the size of φ_i may go up). On the other hand, as computed in Eq. 3, the basic transmission time of each packet also increases. Both effects can cause a growth of the i-level busy-period, which may involve more interference to f_i as well. Therefore, the schedulability ratio decreases when the path lengths go up.

Moreover, when the paths are short, we can observe that EDBT and HPDBT have very close performance. However, as the path lengths go up, the drawback of EDBT becomes more obvious, and HPDBT performs clearly better.

According to the above evaluation results, we can clearly observe that the proposed approaches can improve the schedulability ratio compared to the original flit-level preemptive NoC. The improvement achieved by HPDBT is more obvious compared to the improvement achieved by EDBT. Even though we did observe several cases where the flow sets are schedulable with EDBT but unschedulable with HPDBT, such cases occur very rarely.

7.1.2 Simulation based evaluation

In addition to the analysis based evaluation, we have also randomly generated a number of test cases to show how much improvement from HPDBT can be observed in given simulation scenarios. We have developed a cycle-accurate simulator as a proof-of-concept. In the simulator, the processing overhead while a packet traverses each router is considered. The amount of the processing overhead is based on the information provided in Baron (2010). In these tests, we focus on the comparison between the original flit-level preemptive scheduling (i.e. FLP) and the limited-preemptive scheduling using HPDBT.

The results are represented by the *maximum inflation factor* of each flow set. The maximum inflation factor of a flow set shows how much packet sizes can be inflated while all the deadlines are still met. For example, if the inflation factor of a flow set is 1.5, each flow can have a packet size which is at most 50% larger than the original size while keeping the flow set still schedulable. On the other hand, if the inflation factor is 0.8, each flow has to decrease its packet size by at least 20% in order to meet all the deadlines. Therefore, given the same flow set, the scheduling framework which can achieve a higher inflation factor performs better, since this framework is able to afford more real-time traffic.

The network still uses a 8×8 2D meshed topology. 50 flows are generated for each test. The utilization of each flow is randomly selected from the range of [5%, 10%], and the packet size is randomly generated from the range of [100, 300] flits. Each set of experiment is run for at least 2 times of the hyper-period of all the flows (i.e. $2 \times LCM(F)$).

The results of 10 example tests are presented in Table 1. We can observe that HPDBT always achieves a higher inflation factor compared to FLP, no matter in the analysis based results or the simulation based results. The difference between FLP and HPDBT from the analysis results is between 0.01 and 0.11, and the difference from the simulation results varies from 0.01 to 0.23. Since these tests are randomly selected, they may not show the extreme performance of the proposed framework. However, the improvement of HPDBT compared to FLP can be clearly observed.

7.1.3 Case study

A case study is also generated to examine the improvement of HPDBT compared to FLP. The case study is based on an autonomous vehicle application, which has been utilized in Shi et al. (2012), Indrusiak (2014). The application comprises a number of tasks performing different functionalities such as obstacle detection via stereo photogrammetry, navigation control, and stability control. The whole network includes

Table 1 Examples showing the difference between FLP and HPDBT, presented by inflation factors of the flow sets

Test index	FLP		HPDBT	
	Analysis	Simulation	Analysis	Simulation
1	0.89	1.61	0.98	1.75
2	0.95	1.41	1.01	1.49
3	0.98	1.83	1.03	1.96
4	0.96	1.47	1.01	1.58
5	1	1.64	1.11	1.83
6	0.97	1.74	1	1.91
7	1.06	1.59	1.15	1.77
8	0.95	1.67	1.03	1.83
9	1.01	1.29	1.02	1.3
10	0.95	1.85	1.05	2.08

The bold number represents the maximum value of each column

38 real-time flows for inter-task communications. The system is deployed on a 4×4 2D meshed NoC platform with the X–Y routing algorithm. The NoC frequency is set to 100 MHz, and the bandwidth of each link is 3.2 Gbit/s. More details of the flow set parameters and the task mapping can be found in Shi (2009).

In order to observe the difference between our proposed framework and the platform with the original FLP, we apply a stress test on the above application. An additional testing flow is added into the network. The period of the testing flow is equal to the shortest period in the original flow set (i.e. 0.04 s), which implies that the testing flow has the highest priority. The sensitivity test aims to find the maximum packet size of the testing flow such that the whole flow set keeps schedulable. In order to shorten the testing process, we set an inflation factor 45 to the original flow set (i.e. the total utilization of the original flow set increases 45 times). According to the results, using the platform with limited-preemptive scheduling, the testing flow can transmit 2312 more flits every 0.04 s which equates to 226 kb data per second.

7.2 Evaluation for path modification

We have also generated a number of experiments to evaluate the further improvement achieved by the path modification approach introduced in Sect. 6. The considered NoC uses a 4×4 2D-meshed topology, and it contains 50 real-time flows. The utilization of each flow is randomly selected from two ranges, [5, 10%] and [0.03, 10%]. The packet size of each flow is randomly generated from [100, 300] flits. We compare the performance of three frameworks, where one framework uses only HPDBT (without PM), one framework uses HPDBT together with PM (denoted as HPDBT-PM) and one framework uses FLP with PM (denoted as FLP-PM).

In the first set of experiments, the utilization of each flow is generated from [5, 10%], and the results are shown in Fig 11. As the maximum link utilization increases from 45

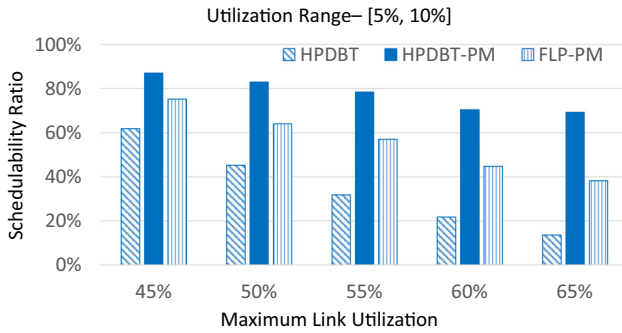


Fig. 11 Comparison between HPDBT, HPDBT-PM and FLP-PM regarding schedulability ratio. Utilization of each flow is selected from [5, 10%]

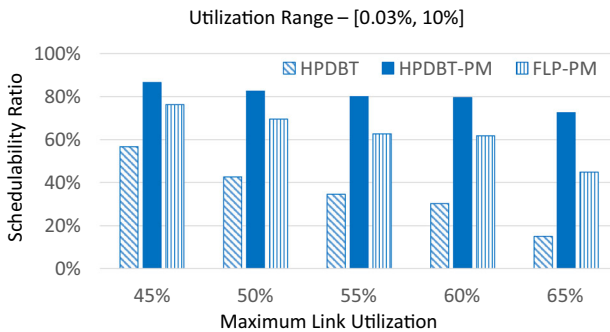


Fig. 12 Comparison between HPDBT, HPDBT-PM and FLP-PM regarding schedulability ratio. Utilization of each flow is selected from [0.03, 10%]

to 65%, the schedulability ratio achieved by HPDBT drops from 61 to 13%, while the schedulability ratio achieved by HPDBT-PM decreases from 87 to 69%. On the other hand, the schedulability ratio of FLP-PM drops from 75 to 38%. We can clearly observe that using path modification can indeed improve the schedulability. The improvement becomes more significant as the maximum link utilization goes up. Moreover, by comparing the results of HPDBT and FLP-PM, we can also observe that the path modification based approach can improve schedulability more effectively compared to the limited preemption based solution. In the second set of experiments, we enlarge the range of flow utilization to [0.03, 10%]. The results are depicted in Fig. 12, where we can obtain a similar observation that using path modification can significantly improve the schedulability.

The above results have already shown the advantage of applying path modification. On the other hand, we realize that the path modification mechanism requires more processing time due to exploring the solution space. Therefore, we have also generated a number of experiments comparing the processing time of HPDBT and HPDBT-PM.

The considered network is 4×4 2D-meshed NoC. The utilization of each flow is randomly selected from [0.03, 10%], and the packet size is randomly generated from [100, 300]. We increase the number of flows in the NoC from 10 to 100 with a step

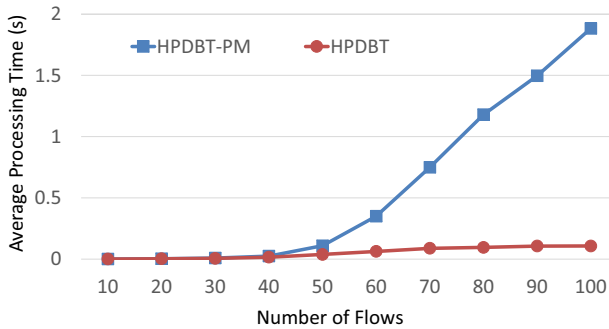


Fig. 13 Comparison between HPDBT and HPDBT-PM regarding average processing time

Table 2 Comparison between HPDBT and HPDBT-PM regarding average processing time.

No. of flows	HPDBT (ms)			HPDBT-PM (ms)		
	Max	Mean	Std.	Max	Mean	Std.
10	10	0.5	2.18	10	0.5	2.18
20	20	2.57	4.51	20	2.57	4.51
30	100	5.69	10.15	110	8.16	12.80
40	170	15.35	17.14	210	24.8	29.63
50	470	37.62	48.51	1500	109.31	181.08
60	1200	62.40	107.70	3051	349.91	458.90
70	790	87.83	113.99	4954	748.93	780.09
80	1811	95.15	157.67	7004	1179.22	1080.54
90	3272	106	255.50	6550	1496.42	1193.23
100	2335	106.97	211.38	8804	1882.99	1303.58

of 10. The results are presented in Fig. 13 and Table 2. According to Fig. 13, as the number of flows goes up, both algorithms require more processing time. However, the average processing time of HPDBT-PM increases much faster than HPDBT. When the number of flows increases from 10 to 100, the processing time of HPDBT raises from less than 0.5–100 ms while the processing time of HPDBT-PM increases from 0.5 to 1.9 s. If we increase the size of NoCs, the required processing time of HPDBT-PM may further increase dramatically because of the larger search space. For example, in a 4×4 NoC, a flow can have at most 20 route candidates. However, in a 6×6 NoC, the maximum number of route candidates for a flow is 252, and this number becomes even 3432 in a 8×8 NoC. Therefore, we can conclude that the path modification approach is an effective means to improve schedulability, however, it can be constrained by the scalability.

8 Conclusion and future works

In this paper, we present a new scheduling framework of wormhole-switched NoCs with VCs, where non-preemptive regions are introduced to NoC packets. The proposed

solution aims to improve the schedulability of the whole network. A schedulability test of the proposed framework is provided along with the proof of its correctness. Additionally, we also present a path modification approach based on the above framework in order to further improve schedulability. According to the evaluation results, the proposed approaches can always achieve higher schedulability ratios compared to the original flit-level preemptive NoC. As a future work, the framework can be extended to support priority sharing policies, so that the required number of VCs can be reduced. Moreover, we also plan to apply the proposed approaches on NoCs with arbitrary buffer sizes.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Baron M (2010) The single-chip cloud computer—intel networks 48 pentiums on a chip. Intel
- Benini L, De Micheli G (2002) Networks on chips: a new soc paradigm. *Computer*
- Bertogna M, Buttazzo G, Yao G (2011a) Improving feasibility of fixed priority tasks using non-preemptive regions. In: 32nd real-time systems symposium (RTSS), IEEE
- Bertogna M, Khani O, Marinoni M, Esposito F, Buttazzo G (2011b) Optimal selection of preemption points to minimize preemption overhead. In: 23rd Euromicro conference on real-time systems (ECRTS), IEEE
- Bril RJ, Lukkien JJ, Verhaegh WE (2009) Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Syst* 42(1):63–119
- Bril RJ, Van den Heuvel MM, Keskin U, Lukkien JJ (2012) Generalized fixed-priority scheduling with limited preemptions. In: 24th Euromicro conference on real-time systems (ECRTS), IEEE
- Cota É, de Morais Amory A, Lubaszewski MS (2011) Reliability, availability and serviceability of networks-on-chip. Springer, Berlin
- Dally WJ (1992) Virtual-channel flow control. *IEEE Trans Parallel Distrib Syst* 3(2):194–205
- Dasari D, Nikolić B, Nélis V, Petters SM (2014) Noc contention analysis using a branch-and-prune algorithm. *TECS* 13:113
- Davis R, Bertogna M (2012) Optimal fixed priority scheduling with deferred pre-emption. In: 33rd real-time systems symposium (RTSS), IEEE
- De Dinechin BD, Durand Y, Van Amstel D, Ghiti A (2014) Guaranteed services of the NoC of a manycore processor. In: 7th international workshop on network on chip architectures (NoCArc), ACM
- Diemer J, Ernst R (2010) Back suction: service guarantees for latency-sensitive on-chip networks. In: 4th ACM/IEEE international symposium on networks-on-chip (NoCs), IEEE
- Ferrandiz T, Frances F, Fraboul C (2009) A method of computation for worst-case delay analysis on spacewire networks. In: 4th international symposium on industrial embedded systems (SIES), IEEE
- Goossens K, Dielissen J, Radulescu A (2005) *Æthereal* network on chip: concepts, architectures, and implementations. *Des Test Comput* 22:414–421
- Indrusiak LS (2014) End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *J Syst Arch* 60(7):553–561
- Joseph M, Pandya P (1986) Finding response times in a real-time system. *Comput J* 29:390–395
- Kashif H, Gholamian S, Patel H (2015) SLA: a stage-level latency analysis for real-time communication in a pipelined resource model. *IEEE Trans Comput* 64(4):1177–1190
- Lehoczy JP (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: 11th real-time systems symposium (RTSS)
- Liu CI, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. *J ACM* 20:46–61

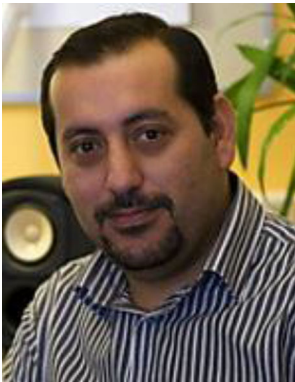
- Liu M, Becker M, Behnam M, Nolte T (2016) Scheduling real-time packets with non-preemptive regions on priority-based NoCs. In: 22nd international conference on embedded and real-time computing systems and applications (RTCSA), IEEE
- Ni LM, McKinley PK (1993) A survey of wormhole routing techniques in direct networks. *Computer* 26(2):62–76
- Nikolic B, Pinho LM, Indrusiak LS (2016) On routing flexibility of wormhole-switched priority-preemptive NoCs. In: 22nd international conference on embedded and real-time computing systems and applications (RTCSA), IEEE
- Paukovits C, Kopetz H (2008) Concepts of switching in the time-triggered network-on-chip. In: 14th international conference on embedded and real-time computing systems and applications (RTCSA), IEEE
- Saksena M, Wang Y (2000) Scalable real-time system design using preemption thresholds. In: 21st real-time systems symposium (RTSS), IEEE
- Shi Z (2009) Real-time communication services for networks on chip. PhD thesis, University of York
- Shi Z, Burns A (2008) Real-time communication analysis for on-chip networks with wormhole switching. In: 2nd international symposium on networks-on-chip NoCs, ACM/IEEE
- Shi Z, Burns A (2009a) Improvement of schedulability analysis with a priority share policy in on-chip networks. In: 17th international conference on real-time and network systems (RTNS)
- Shi Z, Burns A (2009b) Real-time communication analysis with a priority share policy in on-chip networks. In: 21st Euromicro conference on real-time systems (ECRTS), IEEE
- Shi Z, Burns A, Indrusiak LS (2012) Schedulability analysis for real time on-chip communication with wormhole switching. In: Innovations in embedded and real-time systems engineering for communication
- Song H, Kwon B, Yoon H (1997) Throttle and preempt: a new flow control for real-time communications in wormhole networks. In: 26th international conference on parallel processing (ICPP), IEEE
- Wang Y, Saksena M (1999) Scheduling fixed-priority tasks with preemption threshold. In: 6th international conference on real-time computing systems and applications (RTCSA), IEEE
- Wentzlaff D, Griffin P, Hoffmann H, Bao L, Edwards B, Ramey C, Mattina M, Miao C, Brown JE III, Agarwal A (2007) On-chip interconnection architecture of the tile processor. *IEEE Micro* 27:15–31
- Xiong Q, Lu Z, Wu F, Xie C (2016) Real-time analysis for wormhole NoC: revisited and revised. In: 26th edition on great lakes symposium on VLSI, ACM



Meng Liu is a PhD student at Mälardalen University since September, 2012. He received his B.Sc. degree in Network Engineering from Tianjin Polytechnic University, China in 2009. In the year 2011, he got his Master degree in Network and Distributed Systems from Chalmers University of Technology, Sweden. After one years working as software developer, he started his PhD studies in Mälardalen University. He is a member of the Complex Real-Time Embedded Systems (CORE) group at Mälardalen Real-Time Research Center (MRTC). His research interests lie in real-time and embedded systems, including real-time communication, network-on-chip, scheduling and time analysis, stochastic and statistical analysis.



Matthias Becker is a PhD student at Mälardalen University since October 2013. He received his B.Eng. degree in Mechatronics/Automation Systems from the University of Applied Sciences Esslingen, Germany in 2011. In the year 2013 he got his M.Sc. degree in Computer Science specializing in embedded computing from the University of Applied Sciences Munich, Germany. He has been a visiting researcher at CISTER - Research Centre in Real-Time and Embedded Computing Systems in Porto in 2015 and 2016. He is a member of the Complex Real-Time Embedded Systems (CORE) group at Mälardalen Real-Time Research Center (MRTC). His research is in the field of many-core real-time systems with particular interest in predictable execution frameworks for industrial systems.



Moris Behnam was awarded a B.Eng., and M.Sc. in Computer and Control Engineering at the University of Technology, Iraq, and also M.Sc., Licentiate, and PhD in Computer Science and Engineering at MDH, Sweden, in 1995, 1998, 2005, 2008 and 2010 respectively. He has been a visiting researcher at Wayne State University, USA in 2009 and he has been a Postdoctoral Researcher at University of Porto in 2011. His research interests include real-time scheduling, synchronization protocols, multicore/multiprocessor systems, distributed embedded real-time systems and using control theories in real-time scheduling.



Thomas Nolte was awarded a B.Eng., M.Sc., Licentiate, and Ph.D. degree in Computer Engineering from Mälardalen University (MDH), Västerås, Sweden, in 2001, 2002, 2003, and 2006, respectively. He has been a Visiting Researcher at University of California, Irvine (UCI), Los Angeles, USA, in 2002, and a Visiting Researcher at University of Catania, Italy, in 2005. He has been a Postdoctoral Researcher at University of Catania in 2006, and at MDH in 2006–2007. Thomas Nolte became an Assistant Professor at MDH in 2008, and Associate Professor at MDH in 2009. 2012 he became Full Professor of Computer Science. His research interests include predictable execution and performance of embedded systems, design, modeling and analysis of real-time systems, multicore systems, distributed embedded real-time systems, internet-of-things and the cloud.