
Article:
Nikolic, Borislav, Tobuschat, Sebastian, Soares Indrusiak, Leandro et al. (2019) Real-Time Analysis of Priority-Preemptive NoCs with Arbitrary Buffer Sizes and Router Delays. Real-Time Systems. pp. 63-105. ISSN: 1573-1383

https://doi.org/10.1007/s11241-018-9312-0

Noname manuscript No.
(will be inserted by the editor)

Real-Time Analysis of Priority-Preemptive NoCs with Arbitrary Buffer Sizes and Router Delays

Borislav Nikolić[1] · **Sebastian Tobuschat**[1] · **Leandro Soares Indrusiak**[2] · **Rolf Ernst**[1] · **Alan Burns**[2]

Received: date / Accepted: date

Abstract Nowadays available multiprocessor platforms predominantly use a Network-on-Chip (NoC) architecture as an interconnect medium, due to its good scalability and performance. During the last decade, NoCs received a significant amount of attention from the real-time community. One promising category of approaches suggests to employ already existing hardware features called virtual channels, and dedicate them, exclusively, to individual communication traffic flows. In this way, NoCs become more amenable to the real-time analysis, which is an essential requirement for providing both safe and tight worst-case analysis methods, and consequently deriving real-time guarantees.

In this manuscript, we present the approach which falls in the aforementioned category. Specifically, we propose a novel method for the worst-case analysis of the NoC traffic, assuming the existence of per-flow dedicated virtual channels. Compared to the state-of-the-art techniques, our approach yields substantially tighter upper-bounds on the worst-case traversal times (WCTTs) of communication traffic flows. By employing the proposed method, resource over-provisioning can be mitigated to a large extent, and significant design-cost reductions can be achieved. Moreover, we implemented a cycle-accurate simulator of the assumed NoC architecture, and used it to assess the tightness of derived WCTT bounds. Finally, we reached an interesting conclusion that bigger virtual channel buffers do not necessarily lead to better results, and in many cases can be counter-productive, which is a very important finding for system designers.

Keywords Real-Time Systems · Embedded Systems · Network-on-Chip · Wormhole Switching · Virtual Channels · Priority-Preemptive Arbitration

[1] Institute of Computer and Network Engineering, Technische Universität Braunschweig, Germany

[2] Real-Time Systems Group, Department of Computer Science, University of York, York, UK

Borislav Nikolić(✉)
bnikolic@ida.ing.tu-bs.de

1 Introduction

The Network-on-Chip (NoC) architecture (Benini and De Micheli (2002)) is the predominant choice for interconnect mediums in nowadays available multiprocessor platforms. The popularity of NoCs can be largely attributed to their good performance and scalability potential (Kavaldjiev and Smit (2003)). NoCs can considerably vary in terms of various design-choices. One example is the network topology. Currently available multiprocessors employ a *ring* (e.g. Intel (2013)), a *2-D torus* (e.g. Kalray (2014)) and a *2-D mesh* (e.g Tilera (2012) and Intel (2010)) approach. Moreover, NoCs employ different switching mechanisms. For example, a *store-and-forward* technique is a viable strategy, although more popular for off-chip networks than for NoCs. A more promising mechanism is the *wormhole* switching technique (Ni and McKinley (1993)), due to its good throughput and small buffering requirements (Kavaldjiev and Smit (2003)). With this method, the communication packet is, prior to sending, divided into small elements of fixed size, called *flits*. Flits are sequentially injected into the NoC, and they travel in parallel, which is called the *pipelined* traversal. The first flit is called the *header* flit, and it usually contains the relevant information for the traversal of the packet across the NoC.

Another important design choice for NoCs is the routing mechanism. Of interest for real-time systems are static routing algorithms, of which the most popular one is the dimension-ordered routing method called the *X-Y routing* technique. With this approach, all flits constituting one packet first travel on the X-axis, and once they reach the X coordinate of the destination, the transfer continues on the Y-axis. This method is very appreciated by both the academia and industry, primarily because of its relatively easy implementation and a deadlock-free property (Hu and Marculescu (2003)). However, recent insights (e.g. Nikolić et al (2016b) and Nikolić and Pinho (2017)) suggest that this method may not be the most efficient routing approach. Some alternative strategies are to encode the entire path of the packet inside the header flit (e.g. Kalray (2014)), or to preconfigure routers with the relevant routing information (e.g. Stefan et al (2012)).

Yet another design choice is the flow-control strategy. This aspect is very important, because its main purpose is to prevent buffer overflows and packet drops. One of the most prominent approaches is the *credit-based* flow control, which allows a flit transfer only if there are available credits in a given router. Initially, all routers have credits. The amount of credits corresponds to the available space in buffers. Each flit transfer (downstream) is followed by a credit transfer in the opposite direction (upstream). Some alternative flow-control mechanisms are *back suction* (Diemer and Ernst (2010)) and a source router traffic shaping (Kalray (2014)), while some architectures do not use any flow control mechanism and buffer overflows are prevented by design (e.g. Schoeberl et al (2015)).

NoCs can also vary in terms of employed arbitration mechanisms. When several packets compete for some shared NoC resource (e.g. a common output link), these techniques organise accesses. One of the mainstream options is the

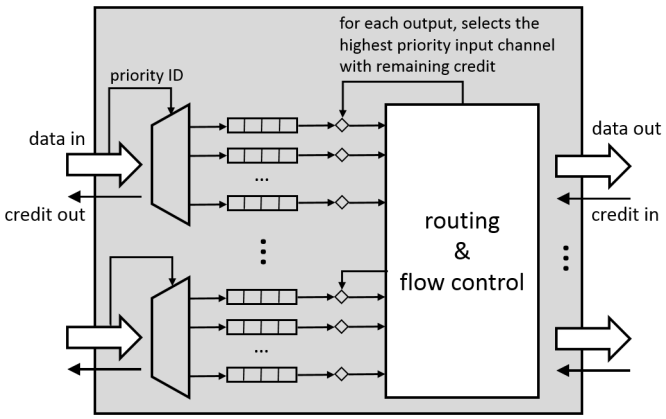


Fig. 1 Router with priority-preemptive arbitration and credit-based flow control

round robin mechanism. Alternative approaches, which are also more real-time oriented, advocate prioritisation, where packets can have static (e.g. Indrusiak et al (2016)) or dynamic (e.g. Nikolić and Petters (2014a)) priorities.

Finally, NoCs can differ in terms of additional hardware features, such as *virtual channels* (Dally (1992), Dally and Seitz (1987)). A virtual channel is nothing more than a buffer dedicated to a given port of a given router. Virtual channels allow to simultaneously buffer flits from different traffic flows. This can significantly mitigate negative effects of some infamous contention scenarios which may cause severe performance deterioration, e.g. *head of line blocking* (Dally (1992)). Another benefit of virtual channels is that when two packets compete for the same resource, the higher-priority one can be granted the permission to progress, while the lower-priority one can be stored inside its virtual channel and in that way delayed until the next arbitration event (Song et al (1997)). This gives the possibility to enforce priority-preemptive strategies for NoCs (Shi and Burns (2008b)). Such a router architecture is illustrated in Figure 1. This type of NoCs is amenable to the real-time analysis, because the worst-case analysis methods can be efficiently applied to this model. Due to these reasons, the priority-preemptive NoCs are currently considered to be a promising approach for the interconnect medium in the forthcoming generation of real-time oriented multiprocessors.

Contribution: In this manuscript, we present a novel analysis method for computing the worst-case traversal times (WCTTs) of communication traffic flows. The proposed technique is applicable to workloads deployed upon priority-preemptive NoCs with the wormhole switching mechanism and per-traffic flow dedicated virtual channels. Compared to the state-of-the-art approaches, the proposed method obtains significantly tighter upper-bounds. This aspect is very important during the design phase of real-time systems, because it allows to mitigate over-provisioning and achieve substantial design-cost reductions. Moreover, we implemented a cycle-accurate simulator of the assumed NoC architecture, and used it to assess the tightness of derived upper-

bounds on WCTTs of traffic flows. Finally, the experimental evaluation led us to an interesting finding that bigger virtual channel buffers do not necessarily yield better results, and in many cases can be counter-productive, which is a very important discovery for system designers.

2 Related Work

In the real-time analysis of NoCs, there are two different strategies. One category of approaches advocates to do a design-time temporal and/or spatial allocation of NoC resources to given communication traffic-flows. In this way, any contentions for shared resources can be avoided, and these methods are called *contentionless approaches*. One popular strategy to achieve this is by arbitrating the access to the NoC and its resources in a time-division-multiplexing (TDM) manner, while some others revolve around reserving all resources on the entire path of the flow prior to its release, often called the *virtual circuit* method. Some notable approaches have been proposed by Millberg et al (2004), Goossens et al (2005), Schoeberl (2007), Paukovits and Kopetz (2008), Stefan et al (2012), Schoeberl et al (2015), Kasapaki et al (2016).

On the other hand, there are methods which allow contentions among traffic flows, termed *contention-aware approaches*. For NoCs with round-robin arbitration some relevant works have been developed around the *recursive calculus* (e.g. Dasari et al (2013), Liu et al (2017)) and the *network calculus* theory (e.g. Ferrandiz et al (2011), de Dinechin et al (2014a), de Dinechin et al (2014b)).

In scenarios where a NoC has multiple virtual channels, preemptions among traffic flows can be performed (Song et al (1997)). Shi and Burns (2008b) developed this approach further, employed several additional assumptions (constrained deadlines, distinctive priorities, per-priority virtual channels) are proposed the analysis method for computing WCTTs of traffic flows. Shi et al (2010) then extended the method and made it applicable to flow-sets with arbitrary deadlines. Nikolić et al (2013) reduced hardware requirements of this model by demonstrating that with a thoughtful allocation of virtual channels, their number can be reduced from the total number of priorities (flows), to the maximum number of contentions for any port. Regarding the same model, Shi and Burns (2008a) and Liu et al (2015a) developed heuristic-based exhaustive search methods for priority assignment. Nikolić and Petters (2014a) proposed an arbitration policy based on the earliest-deadline-first (EDF) methodology, and derived the accompanying WCTT analysis method for flows. Nikolić et al (2016b) and Nikolić and Pinho (2017) explore the routing flexibility of priority-preemptive NoCs, and propose a method to derive flow routes, which allows to utilise platform resources more efficiently than the X-Y routing policy. Liu et al (2015b) focus on the stochastic response time analysis. Moreover, Nikolić et al (2016a) and Liu et al (2016b) discovered that the analysis pessimism can be reduced by considering only parts of paths shared by interfering flows.

The area of workload mapping has also been extensively studied, with the main incentive to map the workload in such a way that existing resources can be utilised more efficiently, and consequently additional traffic flows can be accommodated. Shi and Burns (2010) use a task swapping strategy, Mesidis and Indrusiak (2011) and Racu and Indrusiak (2012) employ the genetic algorithms metaheuristic, while Nikolić et al (2013) focus on the simulated annealing metaheuristic. Sayuti and Indrusiak (2013) study the mapping process with an emphasis on the NoC power consumption, while Nikolić and Petters (2014b) proposed a heuristic-based application mapping approach for the Limited Migrative Model (LMM).

In terms of joint computation and communication guarantees (also called *end-to-end guarantees*), Indrusiak (2014) proposed a schedulability analysis method for a fully-partitioned many-core system, while Nikolić et al (2014) derived the worst-case analysis approach for LMM. Additionally, Burns et al (2014) and Indrusiak et al (2015) demonstrated that priority-preemptive NoCs can accommodate the mixed-criticality workload.

All the aforementioned approaches rely on the assumption that, during the worst-case analysis, entire flow routes are treated as indivisible resources. Kashif et al (2014) proposed a different approach called SLA (*stage level analysis*), where the worst-case analysis is performed iteratively, by considering individual route elements in a sequential manner. One limitation of this method is an unrealistic assumption that virtual channels should have sufficient capacity to store entire packets (Kashif and Patel (2014)). Later, Kashif and Patel (2016) proposed an improved version of SLA which takes into account buffer sizes, and demonstrated that their method derives upper bounds on flow traversal times which are always equal to, or tighter than those produced by the method of Shi and Burns (2008b).

The aforementioned studies consider flit-level preemptions. Recently, Liu et al (2016a) proposed a method for the worst-case analysis assuming limited preemptions via non-preemptive regions. This approach is an initial step towards understanding flow interactions on the packet-level for priority-aware NoCs. Additionally, Shi and Burns (2010) and Liu et al (2016b) analyse priority-preemptive NoCs with shared virtual channels.

Yet another relevant approach is the Compositional Performance Analysis (CPA) method, introduced by Henia et al (2005). Similar to SLA, CPA also applies an iterative approach where network elements are analysed independently. After that, the output events of each element are used as input events for neighbouring elements, and the analysis is performed again. This process is repeated until a converging point is reached (if one exists). Rambo and Ernst (2015) proposed the CPA-based method for the worst-case analysis of priority-preemptive NoCs.

Recently, Xiong et al (2016) discovered that the effect called *backpressure* has a significant impact on the worst-case analysis of flow traversal times, and that it was largely neglected by the community. In fact, their discovery rendered the aforementioned approaches related to priority-premptive NoCs optimistic. One exception is a scenario where virtual channel buffers are large

enough to store entire packets. In this case, backpressure effects are of no significance for the analysis. However, in practical settings, virtual channel buffers have a limited size, and the backpressure effects cannot be neglected. Therefore, Xiong et al (2016) proposed a novel analysis method to compute WCTTs of flows. Subsequently, Indrusiak et al (2016) demonstrated that the aforementioned approach is also optimistic, and proposed a new approach. That work also has a limitation with an unsafe treatment of flows with both upstream and downstream indirect interference. Then, Xiong et al (2017) revised their approach and made it safe. Here, in this work, we refer to that approach as $SOTA$ (short for State-Of-The-Art). Yet, most recently, Indrusiak et al (2018) revised their approach and made it safe, hereafter referred to as $SOTA^+$. Note, that $SOTA^+$ always produces equal or tighter bounds than $SOTA$. Nonetheless, for the sake of completeness of this work, we will compare our approach against both $SOTA$ and $SOTA^+$. A more detailed explanation of these methods is given in Section 5, and for further details the reviewer is advised to consult the work of Indrusiak et al (2018).

Following the aforementioned discoveries, Tobuschat and Ernst (2017) developed a CPA-based worst-case analysis method which takes into account the backpressure effects. However, this is an initial backpressure-aware CPA-based approach, and it is only applicable to NoCs with a single channel. Therefore, it cannot be compared with neither $SOTA$, nor $SOTA^+$, nor the method proposed in this work.

3 System Model

3.1 Platform

The platform θ considered in this work is a multiprocessor system with $m \times n$ processing elements (cores) $\{\pi_1, \pi_2, ..., \pi_{nm}\}$, interconnected via a 2-D mesh NoC. The NoC is composed of $m \times n$ routers $\{\rho_1, \rho_2, ..., \rho_{nm}\}$ (one per core), as illustrated in Figure 2. Note, that in Figure 2 and remaining figures, cores and respective core-to-router input/output links have been omitted for better clarity.

All routers are synchronised, identical, and, depending on its position, each of them is connected with 2, 3 or 4 neighbouring ones. The routing delay is denoted with d_R, and it represents the time that it takes for a header flit to be transferred *within* the router, usually from the input port to the output port. We assume that the header size is equal to the size of one flit. The routing process is typically performed in several pipelined stages. We assume an arbitrary number of pipeline stages, each taking one clock cycle. Note, that remaining flits do not suffer routing delay. Moreover, all flows are routed with the X-Y routing policy.

A connection between each pair of adjacent routers is established via two unidirectional links, while all links of the NoC have identical physical characteristics. The link traversal delay is denoted with d_L, and all flits experience

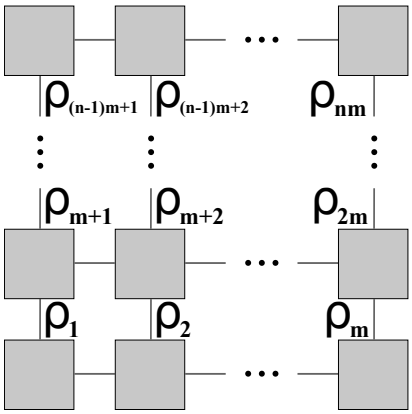


Fig. 2 Assumed NoC interconnect with 2-D mesh topology and $m \times n$ identical routers (for better clarity, cores and respective core-to-router input/output links have been omitted)

the same delay when travelling *between* two adjacent routers. It is assumed that d_L takes one clock cycle. Additionally, each router ρ_i is connected via two unidirectional links with its local core π_i. Core-to-router links are identical to router-to-router links, i.e. a traversal delay of one flit is d_L.

The data transfer is performed with the wormhole switching mechanism and the credit-based flow control is used. It is assumed that the transfer of credits takes less time than the transfer of flits, which is a reasonable assumption, because of the considerable difference in the amount of transferred data. Additionaly, we assume that the platform provides hardware support for data transfer, in the form of virtual channels (VCs), and that the number of VCs is at least equal to the maximum number of contentions for any port of the NoC. This assumption assures that, at each router, each flow will have a dedicated virtual channel (Nikolić et al (2013)). Moreover, we do not put any restrictions of the sizes of individual VC buffers, and each VC can have a capacity to store an arbitrary number of flits. Similarly, we do not put any restrictions on the duration of the routing delay d_R, which makes our approach applicable to a wide range of NoC architectures with the priority-preemprive arbitration.

The assumed router architecture is illustrated in Figure 1. Worth noting is that input ports are connected to output ports via a crossbar and flits from the same input port may travel to different output ports in parallel. The arbitration happens at each output port, and consequently, flits from the highest priority input buffer with credits are transferred to the downstream router. This implies that two flows may interfere only if they share the same output port.

3.2 Workload

In this work, we take a communication-centric approach, and assume that a workload is comprised of a collection of sporadic communication traffic flows

$\mathcal{F} = \{f_1, f_2, ...f_z\}$, hereafter also referred to as the *flow-set*. Each flow $f_i \in \mathcal{F}$ is a source of a potentially infinite sequence of packets, and it has the following characteristics: (i) its source core/router π_i^{src}/ρ_i^{src}, (ii) its destination core/router π_i^{dst}/ρ_i^{dst}, (iii) a set of traversed links between the source and the destination (including the one connecting π_i^{src} with ρ_i^{src}, and the one connecting ρ_i^{dst} with π_i^{dst}), termed $\mathcal{L}_i$, and also called the flow path, which conforms with the X-Y routing policy, (iv) its size σ_i (including the header), expressed in the number of flits, (v) its minimum inter-arrival time T_i between two successive packets, (vi) its constrained deadline $D_i \leq T_i$, (vii) a unique fixed priority $P_i \in \{1, 2, ...|\mathcal{F}|\}$, where $|\mathcal{F}|$ symbolises the number of flows in the flow-set, also called the cardinality of $\mathcal{F}$, and finally (viii) a release jitter J_i^R.

During each inter-arrival period, a flow may release one packet. Let R_i^* be the observed WCTT of any packet of f_i, e.g. via simulations. If R_i^* is less than D_i, it can be conjectured that f_i will never miss its deadline. However, this cannot be guaranteed, unless extensive simulations are performed, so as to capture all possible system states. This can be prohibitively expensive both in terms of computational capacities and time. In this work, we take a different approach and derive an analytical upper bound on the WCTT of flow f_i, termed R_i. Thus, if we prove that derived R_i is a safe upper bound on the WCTT of f_i and that $R_i \leq D_i$, we also proved that f_i will never miss its deadline. If a flow never misses its deadline, it is considered *schedulable*. If all flows of the flow-set are schedulable, the flow-set itself is considered *schedulable*.

4 Problem Formulation

The research problem which is tackled in this work can be summarised as follows. Given the platform θ and the workload $\mathcal{F}$, provide an analysis method to examine the schedulability of $\mathcal{F}$ on θ, by obtaining safe and tight upper bounds on the worst-case traversal times of all flows from $\mathcal{F}$.

5 Background and Preliminaries

We start this section by summarising the aspects of interest in Table 1. Variable $\mathcal{L}_{i,j}^{CD}$ denotes the set of shared links between flows f_i and f_j, which is hereafter referred to as the *contention domain*. Similarly, $\mathcal{L}_{i,j}^{PRE}$ and $\mathcal{L}_{i,j}^{POST}$ denote sets of links on the path of f_i before and after its contention domain with f_j, respectively.

Now we define the traversal delay of flow f_i in isolation, also called the *basic network latency*, or the *zero-load latency* (Equation 1).

$$C_i = \overbrace{(h_i - 1) \cdot d_R}^{\text{header routing}} + \overbrace{h_i \cdot d_L}^{\text{header traversal}} + \overbrace{(\sigma_i - 1) \cdot d_L}^{\text{payload traversal}} \quad (1)$$

Table 1 Platform and flow characteristics

β	VC buffer size (in flits)
d_R	Routing delay of a header flit
d_L	Link traversal delay of one flit
$\mathcal{L}_i$	Set of links traversed by f_i
h_i	Number of elements in $\mathcal{L}_i$ (hops of f_i)
$\alpha(\mathcal{L}_i)$	The first element (link) of $\mathcal{L}_i$
$\omega(\mathcal{L}_i)$	The last element (link) of $\mathcal{L}_i$
$order(\ell_i, \mathcal{L}_j)$ e.g.	The order of link ℓ_i on $\mathcal{L}_j$ $order(\alpha(\mathcal{L}_i), \mathcal{L}_i) = 1$ and $order(\omega(\mathcal{L}_i), \mathcal{L}_i) = h_i$
$\mathcal{L}_{i,j}^{CD}$	$\mathcal{L}_i \cap \mathcal{L}_j$
$\mathcal{L}_{i,j}^{PRE}$	$\bigcup \ell_k \in \mathcal{L}_i \mid order(\ell_k, \mathcal{L}_i) < order(\alpha(\mathcal{L}_{i,j}^{CD}), \mathcal{L}_i)$
$\mathcal{L}_{i,j}^{POST}$	$\bigcup \ell_k \in \mathcal{L}_i \mid order(\ell_k, \mathcal{L}_i) > order(\omega(\mathcal{L}_{i,j}^{CD}), \mathcal{L}_i)$
P_i	Priority of f_i, where $1 \leq P_i \leq \lvert\mathcal{F}\rvert$ (unique priorities)
T_i	Minimum inter-arrival time of f_i
D_i	Deadline of f_i, where $D_i \leq T_i$
σ_i	Size of f_i, including the header (in flits)
J_i^R	Release jitter of f_i

The first term in Equation 1 is the cumulative routing delay of the header flit in all routers from ρ_i^{src} to ρ_i^{dst} (including them). The second term is the cumulative traversal delay of the header flit across all links (including the one connecting π_i^{src} with ρ_i^{src}, and the one connecting ρ_i^{dst} with π_i^{dst}). The last term is the delay of the traversal of remaining flits across the last link, due to the pipelined transmission.

Equation 1 is often simplified with the $d_L = d_R = 1$ assumption, and then it becomes Equation 2.

$$C_i = 2 \cdot h_i + \sigma_i - 2 \tag{2}$$

An even further common simplification is $d_L = 1$ and $d_R = 0$, and then Equation 1 becomes Equation 3.

$$C_i = h_i + \sigma_i - 1 \tag{3}$$

Definition 1 (Basic network latency) Regardless of the computation way (Equations 1-3), C_i is referred to as the basic network latency, or the isolation latency of f_i.

Due to synchronised routers and the pipelined manner of the routing process, as well as the fact that events occur with the granularity of one cycle, the

analysed flow cannot suffer any interference from lower priority flows. However, it can be preempted by some higher priority flows. Therefore, we need to define them.

Definition 2 (Directly interfering flow) Flow f_j is a directly interfering flow of flow f_i *iff* (if and only if) $P_j > P_i$ and $\mathcal{L}_{i,j}^{CD} \neq \emptyset$.

$\mathcal{F}_D(f_i)$ denotes the set of all directly interfering flows of f_i. The interference that analysed flow f_i might suffer from $f_j \in \mathcal{F}_D(f_i)$ in general case does not depend only on f_j, but also on its own directly interfering flows. Therefore, it is necessary to perform an even further classification of flows from $\mathcal{F}_D(f_i)$.

Definition 3 (Directly interfering flow without indirect interference) Flow f_j is a directly interfering flow of flow f_i without indirect interference *iff* all directly interfering flows of f_j are also directly interfering flows of f_i.

$\mathcal{F}_D^{\emptyset}(f_i)$ denotes the set of all directly interfering flows of f_i without indirect interference. Each $f_j \in \mathcal{F}_D^{\emptyset}(f_i)$ we call *difo* of f_i.

Definition 4 (Directly interfering flow with only upstream indirect interference) Flow f_j is a directly interfering flow of flow f_i with only upstream indirect interference *iff* the following two conditions are fulfilled:

- There exists at least one flow f_k which is a directly interfering flow of f_j, but not of f_i, and the contention domain of f_j and f_k is located on the path of f_j upstream from the contention domain of f_j and f_i.
- There exists no flow f_m which is a directly interfering flow of f_j, but not of f_i, and the contention domain of f_j and f_m is located on the path of f_j downstream from the contention domain of f_j and f_i.

$\mathcal{F}_D^{U}(f_i)$ denotes the set of all directly interfering flows of f_i with only upstream indirect interference. Each $f_j \in \mathcal{F}_D^{U}(f_i)$ we call *difu* of f_i.

Definition 5 (Directly interfering flow with only downstream indirect interference) Flow f_j is a directly interfering flow of flow f_i with only downstream indirect interference *iff* the following two conditions are fulfilled:

- There exists at least one flow f_k which is a directly interfering flow of f_j, but not of f_i, and the contention domain of f_j and f_k is located on the path of f_j downstream from the contention domain of f_j and f_i.
- There exists no flow f_m which is a directly interfering flow of f_j, but not of f_i, and the contention domain of f_j and f_m is located on the path of f_j upstream from the contention domain of f_j and f_i.

$\mathcal{F}_D^{D}(f_i)$ denotes the set of all directly interfering flows of f_i with only downstream indirect interference. Each $f_j \in \mathcal{F}_D^{D}(f_i)$ we call *difd* of f_i.

Definition 6 (Directly interfering flow with both upstream and downstream indirect interference) Flow f_j is a directly interfering flow of flow f_i with both upstream and downstream indirect interference *iff* the following two conditions are fulfilled:

– There exists at least one flow f_k which is a directly interfering flow of f_j, but not of f_i, and the contention domain of f_j and f_k is located on the path of f_j downstream from the contention domain of f_j and f_i.
– There exists at least one flow f_m which is a directly interfering flow of f_j, but not of f_i, and the contention domain of f_j and f_m is located on the path of f_j upstream from the contention domain of f_j and f_i.

$\mathcal{F}_D^{U+D}(f_i)$ denotes the set of all directly interfering flows of f_i with both upstream and downstream indirect interference. Each $f_j \in \mathcal{F}_D^{U+D}(f_i)$ we call *difud* of f_i.

Flow relations are formally summarised in Table 2.

Table 2 Flow relations

$\mathcal{F}_D(f_i)$	$\forall f_j \in \mathcal{F} \mid P_j > P_i \wedge \mathcal{L}_i \cap \mathcal{L}_j \neq \emptyset$
$\mathcal{F}_D^{\emptyset}(f_i)$	$\forall f_j \in \mathcal{F}_D(f_i) \mid \mathcal{F}_D(f_j) \setminus \mathcal{F}_D(f_i) = \emptyset$
$\mathcal{F}_D^{U}(f_i)$	$\forall f_j \in \mathcal{F}_D(f_i), \exists f_k \in \mathcal{F}_D(f_j), \nexists f_m \in \mathcal{F}_D(f_j) \mid$ $order(\omega(\mathcal{L}_{j,k}^{CD}), \mathcal{L}_j) < order(\alpha(\mathcal{L}_{j,i}^{CD}), \mathcal{L}_j)$ $\wedge$ $order(\alpha(\mathcal{L}_{j,m}^{CD}), \mathcal{L}_j) > order(\omega(\mathcal{L}_{j,i}^{CD}), \mathcal{L}_j)$
$\mathcal{F}_D^{D}(f_i)$	$\forall f_j \in \mathcal{F}_D(f_i), \exists f_k \in \mathcal{F}_D(f_j), \nexists f_m \in \mathcal{F}_D(f_j) \mid$ $order(\alpha(\mathcal{L}_{j,k}^{CD}), \mathcal{L}_j) > order(\omega(\mathcal{L}_{j,i}^{CD}), \mathcal{L}_j)$ $\wedge$ $order(\omega(\mathcal{L}_{j,m}^{CD}), \mathcal{L}_j) < order(\alpha(\mathcal{L}_{j,i}^{CD}), \mathcal{L}_j)$
$\mathcal{F}_D^{U+D}(f_i)$	$\forall f_j \in \mathcal{F}_D(f_i), \exists f_k \in \mathcal{F}_D(f_j), \exists f_m \in \mathcal{F}_D(f_j) \mid$ $order(\alpha(\mathcal{L}_{j,k}^{CD}), \mathcal{L}_j) > order(\omega(\mathcal{L}_{j,i}^{CD}), \mathcal{L}_j)$ $\wedge$ $order(\omega(\mathcal{L}_{j,m}^{CD}), \mathcal{L}_j) < order(\alpha(\mathcal{L}_{j,i}^{CD}), \mathcal{L}_j)$

Obviously: $\mathcal{F}_D(f_i) = \mathcal{F}_D^{\emptyset}(f_i) \bigcup \mathcal{F}_D^{U}(f_i) \bigcup \mathcal{F}_D^{D}(f_i) \bigcup \mathcal{F}_D^{U+D}(f_i)$.

In Figure 3 is given an example of flows to demonstrate flow relationships. The interfering sets of flow f_7 are as follows:

$$\mathcal{F}_D(f_7) = \{f_3, f_4, f_5, f_6\},$$
$$\mathcal{F}_D^{\emptyset}(f_7) = \{f_3\}, \quad \mathcal{F}_D^{U}(f_7) = \{f_4\}, \quad \mathcal{F}_D^{D}(f_7) = \{f5\}, \quad \mathcal{F}_D^{U+D}(f_7) = \{f_6\}$$

After defining flow relations, now we present the state-of-the-art analysis methods for obtaining WCTTs, namely $SOTA$ and $SOTA^{+}$. The worst-case traversal time of a flow can be computed by solving Equation 4.

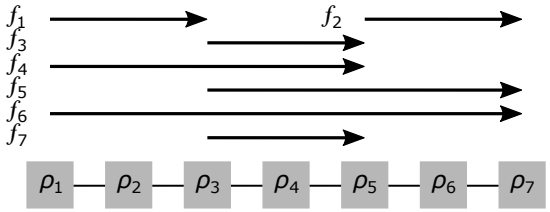


Fig. 3 Example of interfering flows, where $P_1 > P_2 > P_3 > P_4 > P_5 > P_6 > P_7$

$$R_i = C_i + \sum_{f_j \in \mathcal{F}_D(f_i)} \left\lceil \frac{R_i + J_j^R + J_{j\to i}^I}{T_j} \right\rceil \cdot (C_j + B_{j\to i}) \tag{4}$$

In Equation 4, $J_{j\to i}^I$ is the interference jitter, which can be computed by solving Equation 5.

$$J_{j\to i}^I = \begin{cases} R_j - C_j, & \text{if } f_j \in \{\mathcal{F}_D^U(f_i) \bigcup \mathcal{F}_D^D(f_i) \bigcup \mathcal{F}_D^{U+D}(f_i)\} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Both $SOTA$ and $SOTA^+$ use Equation 4 and Equation 5. However, $SOTA$ and $SOTA^+$ differ in the computation of the term $B_{j\to i}$ in Equation 4. The term $B_{j\to i}$ accounts for additional interference that could occur due to *back-pressure*, hereafter referred as the *buffering interference*. In $SOTA$, it can be computed by solving Equation 6.

$$B_{j\to i} = \sum_{\substack{f_k \in \mathcal{F}_D(f_j) \wedge \\ f_k \notin \mathcal{F}_D(f_i) \wedge \\ order(\alpha(\mathcal{L}_{j,k}), \mathcal{L}_j) > \\ order(\omega(\mathcal{L}_{j,i}), \mathcal{L}_j)}} \left\lceil \frac{R_j + J_k^R + J_{k\to j}^I}{T_k} \right\rceil \cdot (C_k + B_{k\to j}) \tag{6}$$

In $SOTA^+$, it can be computed by solving Equation 7.

$$B_{j\to i} = \sum_{\substack{f_k \in \mathcal{F}_D(f_j) \wedge \\ f_k \notin \mathcal{F}_D(f_i) \wedge \\ order(\alpha(\mathcal{L}_{j,k}), \mathcal{L}_j) > \\ order(\omega(\mathcal{L}_{j,i}), \mathcal{L}_j)}} \left\lceil \frac{R_j + J_k^R + J_{k\to j}^I}{T_k} \right\rceil \cdot \begin{cases} min\left\{C_k + B_{k\to j}, \beta \cdot d_L \cdot |\mathcal{L}_{i,j}^{CD}|\right\}, & \text{if } f_j \in \mathcal{F}_D^D(f_i) \\ C_k + B_{k\to j}, & \text{otherwise} \end{cases} \tag{7}$$

From both Equation 6 and Equation 7 we see that the additional buffering interference is caused by each flow f_k which is directly interfering with f_j downstream from the contention domain of f_j and f_i.

There are three limitations of $SOTA$ and $SOTA^+$:

– It is assumed that a flow causes/suffers direct interference during its entire traversal, whereas interference may occur only while interfering/interfered flits traverse the contention domain.

– The buffering interference is unconditionally considered for all *difd* and *difud* flows, while in some cases it may not occur.
– In cases where buffering interference does occur, $SOTA$ and $SOTA^+$ may substantially overestimate it.

All three of these issues are addressed in the next section, and as already mentioned, in this work we will compare the proposed approach against both $SOTA$ and $SOTA^+$.

6 Proposed Approach

In this section, we will present a novel method to compute the WCTT of a traffic flow. First, we will start by analysing only *difo* flows (Section 6.1), and then gradually extend the analysis (Sections 6.2-6.4) until covering all interfering flow categories identified in the previous section.

6.1 Interference from *difo* $\mathcal{F}_D^{\emptyset}$ flows

The common property of all *difo* flows of f_i is that they can suffer interference only from each other, i.e., there are no other flows which can cause interference to them, but not to f_i. This allows us to analyse the system in a very similar way to uniprocessors, where each *difo* flow can be treated as a higher priority task. Shi and Burns (2008b) noticed that in these scenarios indirect interference cannot occur, and therefore interference jitters of all *difo* flows are zero, i.e. $\forall f_j \in \mathcal{F}_D^{\emptyset}(f_i) : J^I_{j\to i} = 0$ (see Equations 4-5).

As already mentioned, both approaches $SOTA$ and $SOTA^+$ consider that flows cause/suffer interference during their entire traversal. First, we will prove that the analysed flow cannot suffer interference from a higher-priority flow while its flits are not within the contention domain (Lemma 1).

Lemma 1 *Consider two flows f_i and f_j, and let $f_j \in \mathcal{F}_D^{\emptyset}(f_i)$. Moreover, let us assume that a packet of f_i was released at the time instant 0. Flow f_i cannot suffer any interference from f_j during the following time intervals: $[0 : \gamma^{PRE}_{i,j}]$, and $[R_i - \gamma^{POST}_{i,j} : R_i]$, where $\gamma^{PRE}_{i,j}$ and $\gamma^{POST}_{i,j}$ can be computed by solving Equation 8 and Equation 9, respectively.*

$$\gamma^{PRE}_{i,j} = \left(\left|\mathcal{L}^{PRE}_{i,j}\right| - 1\right) \cdot d_R + \left|\mathcal{L}^{PRE}_{i,j}\right| \cdot d_L \tag{8}$$

and

$$\gamma^{POST}_{i,j} = \left|\mathcal{L}^{POST}_{i,j}\right| \cdot d_L \tag{9}$$

Proof Proven directly. In order to suffer interference from f_j, flits of f_i need to be inside the respective contention domain. Thus, after its release, it will pass at least $\gamma^{PRE}_{i,j}$ time units before the header of f_i reaches $\alpha(\mathcal{L}^{CD}_{i,j})$ (the first link on the contention domain of f_i and f_j). Therefore, during that time

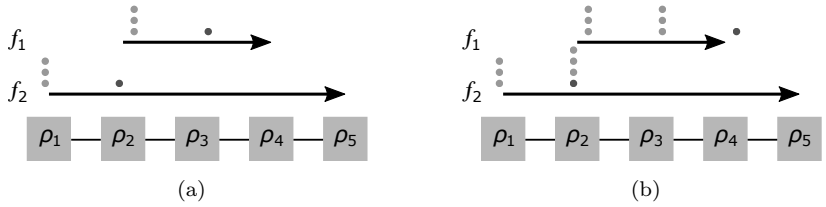


Fig. 4 Detailed analysis of a single preemption, where $P_1 > P_2$, $d_R = 2 \cdot d_L$ and $\beta \geq 4$

f_i cannot suffer interference from f_j. A very similar reasoning can be applied to the last $\gamma_{i,j}^{POST}$ time units after the tail of f_i departs from $\omega(\mathcal{L}_{i,j}^{CD})$ (the last link on the contention domain of f_i and f_j) and reaches the destination. Upon its departure from $\omega(\mathcal{L}_{i,j}^{CD})$, f_i cannot suffer any interference from f_j. □

To summarise, f_i cannot suffer interference from f_j before its header flit reaches the contention domain and after its tail flit departs from the contention domain. Terms $\gamma_{i,j}^{PRE}$ and $\gamma_{i,j}^{POST}$ provide lower bounds on the duration of those intervals, respectively.

Now, let us analyse the interference that a single packet of a higher-priority flow can cause to the analysed packet. Lemma 2 provides an upper bound.

Lemma 2 *Consider two flows f_i and f_j, and let $f_j \in \mathcal{F}_D^{\emptyset}(f_i)$. The maximum interference caused to f_i by a single packet of f_j can be at most $I_{j\to i}$, where $I_{j\to i}$ can be computed by solving Equation 10.*

$$I_{j\to i} = \overbrace{\sigma_j \cdot d_L}^{single\ link\ traversal} + \overbrace{\left(\left|\mathcal{L}_{i,j}^{CD}\right| - 1\right) \cdot min\left\{d_R, \beta \cdot d_L, \sigma_j \cdot d_L\right\}}^{auto\text{-}buffering} \tag{10}$$

Proof Proven directly. Let us analyse the scenario where a single higher-priority packet causes interference to another packet. Figure 4 illustrates such a scenario. Figure 4(a) shows the moment when the header of f_1 started being routed in router ρ_3, and in the next $2 \cdot d_L$ intervals, 2 flits of f_1 arrive in ρ_3. During the next d_L interval, the header finally progresses to ρ_4, while another flit comes to ρ_3 (illustrated in Figure 4(b)).

As observed, f_1 can cause interference to f_2 while all of its flits traverse a single shared link (the first term in Equation 10). Moreover, due to the routing delay, f_1 may cause a self-imposed buffering, hereafter called *auto-buffering*, in each traversed router. Buffered flits are important, because while they are being stalled, e.g. due to the routing of f_1, *already* preempted flits of f_2 may reach those routers, and get preempted *again* when buffered flits start progressing. Thus, buffered flits may cause the interference twice (before being buffered and after being de-buffered), while other flits can cause the interference only once. From the perspective of f_2, the auto-buffering of f_1 matters only inside routers between $\alpha(\mathcal{L}_{1,2}^{CD})$ and $\omega(\mathcal{L}_{1,2}^{CD})$, e.g. only a single router ρ_3 in Figure 4. The autobuffering of f_1 in the shared router before the contention domain need not be considered, because the routing of f_1 in

ρ_2 does not affect the traversal of f_2, and f_1 affects f_2 only when it becomes eligible for transmission (starts competing for a common output link). Also, the autobuffering of f_1 in the shared router after the contention domain (ρ_4) need not be considered, because, f_1 and f_2 do not compete for a common output link any more. Therefore, the maximum auto-buffering interference from f_1 to f_2 is equal to $\left(\left|\mathcal{L}_{i,j}^{CD}\right| - 1\right) \cdot d_R$, which corresponds to the cumulative routing delay of the header flit along the contention domain. If the routing delay inside a single router exceeds the time it takes to fill/empty a corresponding buffer, the auto-buffering interference is then limited by the time it takes to fill/empty the entire buffer $\left(\left|\mathcal{L}_{i,j}^{CD}\right| - 1\right) \cdot \beta \cdot d_L$. Finally, if a packet is so small that it can entirely fit inside a single buffer, the auto-buffering interference is limited by its traversal across the contention domain in a store-and-forward manner $\left(\left|\mathcal{L}_{i,j}^{CD}\right| - 1\right) \cdot \sigma_j \cdot d_L$.

Thus, the total interference that a packet of f_1 can cause to a packet of f_2 can be computed by summing up the interference caused by all flits of f_1 traversing a single link, and adding the auto-buffering interference (Equation 10). □

Note, that Equation 10 is often simplified with the $d_L = d_R = 1$ assumption, and then it becomes Equation 11.

$$I_{j\to i} = \sigma_j + \left|\mathcal{L}_{i,j}^{CD}\right| - 1 \tag{11}$$

An even further common simplification is $d_L = 1$ and $d_R = 0$, and then Equation 10 becomes Equation 12.

$$I_{j\to i} = \sigma_j \tag{12}$$

Also note, that in architectures where, for a given output port, a header routing of one flow and flit transfers of other flows cannot be performed in parallel, Equation 10 becomes Equation 13.

$$I_{j\to i} = \sigma_j \cdot d_L + \left|\mathcal{L}_{i,j}^{CD}\right| \cdot d_R \tag{13}$$

Now, Lemma 3 proves that the findings of Lemma 2 can be extended to three flows.

Lemma 3 *Consider three flows f_i, f_j and f_k, and let $\mathcal{F}_D^{\emptyset}(f_i) = \{f_j, f_k\}$, and $P_k > P_j$. The interference caused to the packet of f_i by single packets of f_j and f_k can be at most $I_{j\to i} + I_{k\to i}$.*

Proof Proven directly. If packets of f_j and f_k cannot interfere with each other (e.g. f_1 and f_2 in Figure 5(a)), or can interfere (e.g. f_1 and f_2 in Figure 5(b)) but traverse at distinctive time intervals, then the results of Lemma 2 apply and the maximum interference that f_3 can suffer is equal to $I_{1\to 3} + I_{2\to 3}$.

Let us consider the case where the higher-priority flows interfere with each other. First, let f_2 cause interference to f_3, while f_1 has not been released yet (Figure 6(a)). Now, let f_1 preempt f_2 and cause its buffering (Figure 6(b)).

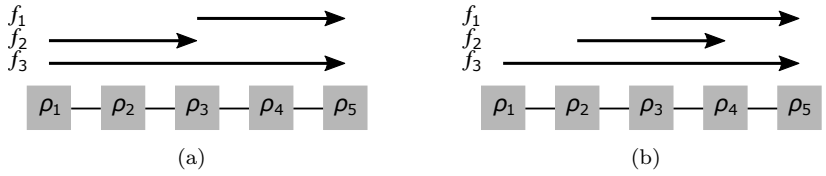


Fig. 5 Example of 3 interfering flows, where $P_1 > P_2 > P_3$

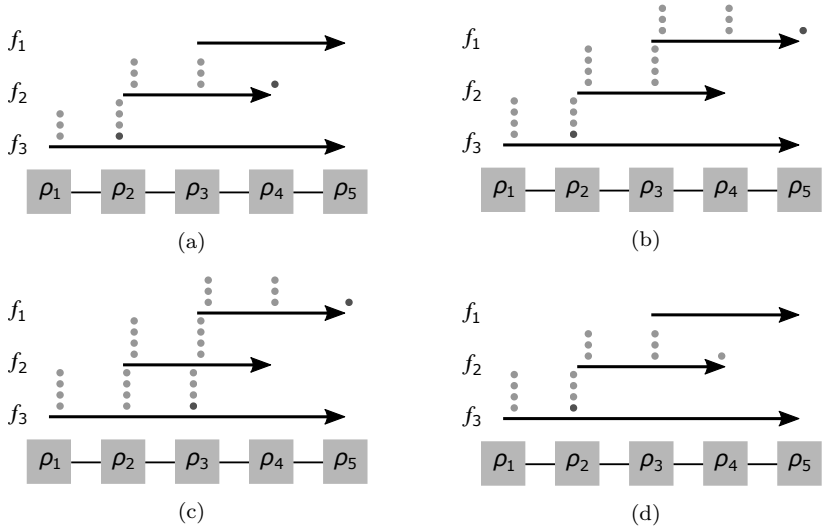


Fig. 6 Detailed contention analysis of 3 flows, where $P_1 > P_2 > P_3$, $d_R = 2 \cdot d_L$ and $\beta = 4$

After f_2 is fully buffered, f_3 is allowed to progress until $\alpha(\mathcal{L}_{1,3}^{CD})$ and then it also gets preempted by f_1 (Figure 6(c)). After f_1 stops interfering with f_2 and f_3 (Figure 6(d)), the former continues its progress, and after that f_3 finally completes its transfer.

Of interest is the interval while f_1 traverses. Notice, that its traversal can cause the preemption and buffering of f_2, where each flit of f_1 can cause the buffering of exactly one flit of f_2 (that is, an interfering flit of f_1 introduces a disruption into the pipeline of f_2 for exactly one flit traversal time d_L). Also notice, that flits of f_1 which cause buffering of f_2 cannot *at the same time* cause interference to f_3, due to (i) the flow positioning, and (ii) the fact that f_1 is progressing and f_2 is being buffered. Thus, every flit of f_1 can either cause a buffering of f_2 or a direct interference to f_3, but not both. This means that if f_1 boosts the buffering interference that f_2 causes to f_3 by x time units, the interference that itself can cause to f_3 is at most $I_{1\to3} - x$. This implies that the maximum interference a packet of f_i can suffer from individual packets of two *difo* flows f_j and f_k is at most the sum of the maximum interferences that they can individually cause. □

In order to generalise the findings of Lemmas 2-3, we have to prove that f_a, which is a *difo* flow of the analysed flow f_i, cannot *generate* interference

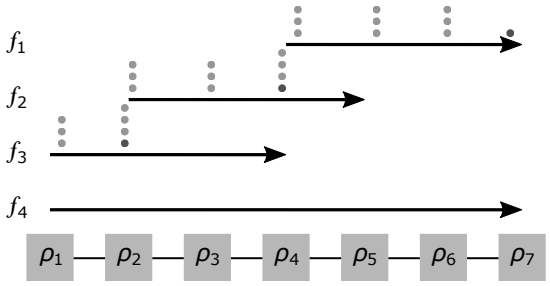


Fig. 7 Example of 4 flows with simultaneous buffering, where $P_1 > P_2 > P_3 > P_4$, $d_R = 2 \cdot d_L$ and $\beta = 4$

greater than $I_{a \to i}$, even though it may cause simultaneous buffering of multiple other *difo* flows of f_i with intermediate priorities. This case is covered with Lemma 4.

Lemma 4 *Consider the flow-set $\mathcal{F} = \{f_1, f_2, ...f_{i-1}, f_i\}$, where $P_1 > P_2 > ... > P_{i-1} > P_i$ and $\mathcal{F}_D^{\emptyset}(f_i) = \{f_1, f_2, ..., f_{i-2}, f_{i-1}\}$. Moreover, consider that each flow releases a single packet. Any higher-priority flow f_a cannot generate interference to f_i (either by preempting, or causing the buffering of other flows) larger than $I_{a \to i}$.*

Proof Proven directly. We claim that, although f_a can cause simultaneous buffering of multiple flows, it cannot induce interference to f_i larger than $I_{a \to i}$. Consider an example illustrated in Figure 7. Notice that f_1 causes the buffering of f_2 which in turn causes the buffering of f_3. Although the traversal of one flit of f_1 causes two buffered flits (one of f_2 and one of f_3), this does not imply that one flit of f_1 generated two flits of interference to f_4. In fact, due to necessary flow positioning to invoke this scenario, as well as the fact that f_2 and f_3 are being simultaneously buffered, flits of f_4 could not suffer interference from flits of f_2 before those flits get buffered (due to f_3 being buffered simultaneously with it), and hence these buffered flits of f_2 can cause interference to f_4 *only once*, while being de-buffered. Thus, buffering of f_2 has no effect on f_4. Therefore, although the traversal of f_1 can cause simultaneous buffering of f_2 and f_3, the buffering of the former is irrelevant for f_4, while the effects of buffering of the latter have been covered with Lemma 3. □

Now we generalise the findings of the aforementioned lemmas with Theorem 1.

Theorem 1 *Consider the flow-set $\mathcal{F} = \{f_1, f_2, ...f_{n-1}, f_n\}$. Let f_i be the analysed flow, and let $\mathcal{F}_D(f_i) = \mathcal{F}_D^{\emptyset}(f_i)$, i.e., f_i has only* difo *flows. The maximum interference that f_i can suffer is at most $I_i^{\emptyset}$, where $I_i^{\emptyset}$ can be computed by solving Equation 14.*

$$I_i^{\emptyset} = \sum_{\forall f_j \in \mathcal{F}_D^{\emptyset}(f_i)} \left\lceil \frac{R_i + J_j^R - \gamma_{i,j}^{PRE} - \gamma_{i,j}^{POST}}{T_j} \right\rceil \cdot I_{j \to i} \tag{14}$$

Proof Proven directly. If each flow emits only a single packet, then the proof straightforwardly follows from Lemmas 1-4. Now, let us analyse the scenario where each flow releases multiple packets. This case can be perceived as if each flow is a set of unrelated same-priority flows, each releasing a single packet. Due to constrained deadlines, if a flow-set is schedulable, at any time instant there will be at most one flow of each priority. Lemmas 1-4 hold for that model, and therefore hold for the model assumed in this work. □

Note, that Equation 14 includes the term R_i, which can be for now considered as a constant, and once we cover all interference scenarios, we will show how to compute it (Equation 27).

6.2 Interference from *difu* $\mathcal{F}_D^U$ flows

Flows with only upstream indirect interference are very similar to the previously covered category of interfering flows. In fact, upstream indirect interference can cause the following three effects, of which only the first two are of relevance for analysed flow f_i:

- Due to upstream indirectly interfering flows, each flow $f_j \in \mathcal{F}_D^U(f_i)$ can appear and preempt f_i *twice* during a time interval which is shorter than T_j. In the literature, this is often called *back-to-back interference*.
- Due to upstream indirectly interfering flows, each flow $f_j \in \mathcal{F}_D^U(f_i)$ can be divided into smaller pieces (sub-packets) that are not mutually pipelined. We call this *packet splitting*.
- Due to upstream indirectly interfering flows, each flow $f_j \in \mathcal{F}_D^U(f_i)$ can be buffered inside routers which are upstream of $\alpha(\mathcal{L}_{i,j}^{CD})$ on the path of f_j, but that has no effect on f_i.

The first effect is taken into account by introducing the interference jitter component $J_{j\to i}^I$, as identified in the initial work of Shi and Burns (2008b). In this work, we will treat the indirect interference in the same way as in all state-of-the-art approaches, by modelling it with the interference jitter (Equation 5).

Now, let us analyse the second effect and check if packet splitting of higher-priority flows affects the analysed flow. This is covered with Lemma 5.

Lemma 5 *Consider two flows f_i and f_j and let $\mathcal{F}_D^U(f_i) = \{f_j\}$. The maximum interference that a single packet of f_j can cause to f_i is the same as if f_j would be a directly interfering flow without the indirect interference, and it is equal to $I_{j\to i}$, where $I_{j\to i}$ can be computed by solving Equation 10.*

Proof Proven directly. The only difference of the packet of f_j from any packet of any flow from $\mathcal{F}_D^\emptyset$ is that it can be split into several pieces (sub-packets). Let us assume that it was split into x packets of equal size σ'_j, that is $\sigma_j = x \cdot \sigma'_j$. Now, let us compute the total interference that these x sub-packets can cause. For clarity purposes, in the following computation we will assume that $d_R < min\{\beta \cdot d_L, \sigma_j \cdot d_L\}$.

$$\sum_{j'=1,\dots,x} I_{j'\to i} = \overbrace{\sigma'_j \cdot d_L + \left(\left|\mathcal{L}^{CD}_{i,j}\right| - 1\right) \cdot d_R}^{\substack{\textit{first sub-packet}\\ \textit{suffers routing overhead}}} + \overbrace{(x-1) \cdot \sigma'_j \cdot d_L}^{\substack{\textit{remaining (x - 1) sub-packets}\\ \textit{do not suffer routing overhead}}} =$$

$$x \cdot \sigma'_j \cdot d_L + \left(\left|\mathcal{L}^{CD}_{i,j}\right| - 1\right) \cdot d_R = \sigma_j \cdot d_L + \left(\left|\mathcal{L}^{CD}_{i,j}\right| - 1\right) \cdot d_R = I_{j\to i}$$

□

Now, we can formulate the maximum interference that the analysed flow can suffer from both *difo* and *difu* flows (Theorem 2).

Theorem 2 *Consider the flow-set $\mathcal{F} = \{f_1, f_2, \dots f_{n-1}, f_n\}$. Let f_i be the analysed flow, and let $\mathcal{F}_D(f_i) = \mathcal{F}^{\emptyset}_D(f_i) \bigcup \mathcal{F}^{U}_D(f_i)$, i.e., f_i has only* difo *and* difu *flows. The maximum interference that f_i can suffer is at most $I^{\emptyset}_i + I^{U}_i$, where $I^{\emptyset}_i$ can be computed by solving Equation 14, and I^{U}_i can be computed by solving Equation 15.*

$$I^U_i = \sum_{\forall f_j \in \mathcal{F}^U_D(f_i)} \left\lceil \frac{R_i + J^R_j + \overbrace{(R_j - C_j)}^{J^I_{j\to i}} - \gamma^{PRE}_{i,j} - \gamma^{POST}_{i,j}}{T_j} \right\rceil \cdot I_{j\to i} \qquad (15)$$

Proof Proven directly. From Lemma 5 it follows that individual packets of *difu* flows can cause the same amount of interference as individual packets of *difo* flows. By applying the existing results on the indirect interference modelling (Shi and Burns (2008b)), we can take into account the indirect interference as the interference jitter (Equation 5). Finally, similar to Theorem 1, we can elevate the conclusions from a packet level to a flow level. □

Note, that in the presence of both *difo* and *difu* flows, if individually observed, *difo* flows may indeed cause more interference to the analysed flow f_i than $I^{\emptyset}_i$, and similarly *difu* flows may cause more interference than I^U_i. This is because of the additional buffering *difus* can cause to *difos* and vice versa. However, the total interference that they can jointly cause to f_i *cannot* exceed the sum of individual terms, as proven by Theorem 2.

6.3 Interference from *difd* $\mathcal{F}^D_D$ flows

Unlike previously covered interfering flow categories, *difd* flows can cause additional interference to the analysed flow. In both approaches $SOTA$ and $SOTA^+$ this additional interference that a *difd* flow f_j can cause to f_i is modelled with the term $B_{j\to i}$ (see Equation 4, Equation 6 and Equation 7). In $SOTA$ and $SOTA^+$ it is always assumed that each *difd* flow will cause

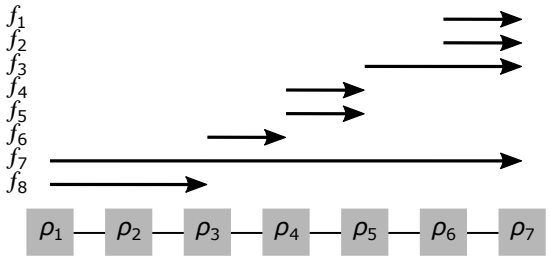


Fig. 8 Example of flows with downstream indirect interference, where $min\{P_1, P_2, P_3, P_4, P_5, P_6\} > P_7 > P_8$

this additional buffering interference, while that may not be always possible. Therefore, in this section we will first derive conditions which are necessary for this additional interference to occur.

6.3.1 Necessary conditions for additional buffering interference

Figure 8 illustrates traffic flows where f_8 may suffer additional buffering interference from flow f_7 due to flows $\{f_1, f_2, f_3, f_4, f_5, f_6\}$. Now, we analyse conditions which cause buffering interference from f_7 to f_8. A starting assumption is that WCTTs of all flows with priorities higher than that of the analysed flow are already computed (in our example $R_1, R_2, R_3, R_4, R_5, R_6$ and R_7). We perform the assessment by executing Algorithm 1 for our example.

Starting from $\omega(\mathcal{L}_{7,8}^{CD})$, get the first link p downstream the path of f_7. Check if there exists flow f_k which traverses p, such that $f_k \in \mathcal{F}_D(f_7) \wedge f_k \notin \mathcal{F}_D(f_8)$ (lines $2-3$ in Algorithm 1). In the example from Figure 8 it is f_6. This flow is added to the set of flows $\overline{\mathcal{F}_D(f_7)}$, which contains all flows interfering with f_7 downstream of $\mathcal{L}_{7,8}^{CD}$ (line 4 in Algorithm 1). If multiple flows fulfil this condition, all are added to the aforementioned set.

Then, the number of intermediate routers is computed (line 6 in Algorithm 1). In our example it is only one router ρ_3. Now, we test if f_7 can be fully buffered inside ρ_3 by evaluating the following condition:

Condition 1: $numRouters \cdot \beta \geq \sigma_7$

If Condition 1 is fulfilled, then f_7 cannot cause buffering interference to f_8 as a consequence of its downstream indirect interference, and therefore it can be treated as a flow with only upstream indirect interference (lines $7-8$ in Algorithm 1). If Condition 1 is not fulfilled, it should be tested whether f_6 can generate enough interference to cause buffering of f_7 all the way upstream to $\mathcal{L}_{7,8}^{CD}$ (lines $11-14$ in Algorithm 1). This is tested with Condition 2:

Condition 2: $numRouters \cdot \beta \cdot d_L \geq \sum_{\forall f_k \in \overline{\mathcal{F}_D(f_7)}} inf(f_k, f_7, p)$,

where $inf(f_k, f_7, p)$ represents the maximum interference that f_k may cause to f_7 on the part of the path currently analysed by Algorithm 1 (until the link p, including it). This value can for now be considered as a constant,

Algorithm 1: BufferingInterferenceExists(f_i, f_j)

```
   input : f_i, f_j
 1 set(F_D(f_j)) = ∅; // Initialise set of flows interfering with f_j downstream of L_{i,j}^{CD}
 2 foreach (p ∈ L_j : order(p, L_j) > order(ω(L_{j,i}^{CD}), L_j)) do
 3 |   foreach (f_k ∈ F_D(f_j) : p ∈ L_k ∧ f_k ∉ F_D(f_i) ∧ f_k ∉ set(F_D(f_j))) do
 4 |   |   add(f_k, set(F_D(f_j))); // Add to flows interfering with f_j downstream of L_{i,j}^{CD}
 5 |   end
 6 |   numRouters = order(p, L_j) − order(ω(L_{j,i}^{CD}), L_j); // Routers for buffering
 7 |   if (numRouters · β ≥ σ_j) then
 8 |   |   return false; // Condition 1 fulfilled, buffering cannot occur
 9 |   else
10 |   |   downstreamInf ← 0; // Variable to compute interference from set(F_D(f_j))
11 |   |   foreach (f_k ∈ set(F_D(f_j))) do
12 |   |   |   downstreamInf ← downstreamInf + inf(f_k, f_j, p);
   |   |   |   // inf(f_k, f_j, p) is the maximum interference from f_k to f_j until p
13 |   |   end
14 |   |   if (numRouters · β · d_L ≥ downstreamInf) then
   |   |   |   // Condition 2 fulfilled, still inconclusive
15 |   |   else
16 |   |   |   return true; // Condition 2 failed, buffering occurs
17 end
18 return false; // Condition 2 always fulfilled, buffering cannot occur
```

and after covering all flow interference scenarios, we will explain how it can be obtained (Equation 26). In our example, $\overline{\mathcal{F}_D(f_7)}$ has only one flow f_6.

If Condition 2 is not fulfilled, then f_7 *can* cause additional buffering interference to f_8 and in the next section we will show how to compute it (line 16 of Algorithm 1). Conversely, if Condition 2 is fulfilled, that means *only* that f_6 alone cannot cause buffering of f_7 to affect f_8. Thus, this is not a sufficient, but only a necessary condition for the absence of buffering interference, and in order to deduce if buffering interference does exist, it is necessary to check further downstream f_7.

The process continues by searching further downstream the path of f_7 for new flows that may cause buffering of f_7 (a new loop at line 2 in Algorithm 1). In our example, there exist 2 new flows, f_4 and f_5. Then, Condition 1 is tested again (for routers ρ_3 and ρ_4), and if it is fulfilled the buffering cannot occur. Otherwise, it is tested whether Condition 2 is fulfilled (this time $\overline{\mathcal{F}_D(f_7)} = \{f_6, f_5, f_4\}$). If it is not fulfilled, the algorithm stops, because f_7 can cause buffering interference to f_8. If it is fulfilled, new iterations are started, covering bigger and bigger portions of the path of f_7. If Algorithm 1 investigates the entire path of f_7 and has Condition 2 always fulfilled, this means that f_7 cannot cause additional buffering interference to f_8 (line 18 in Algorithm 1).

In summary, by iteratively applying Algorithm 1 to all downstream links on the path of a preempting flow, starting with the first link after the contention domain:

- If Condition 1 is fulfilled at least once ⇒ no buffering interference.
- If Condition 2 is always fulfilled ⇒ no buffering interference.

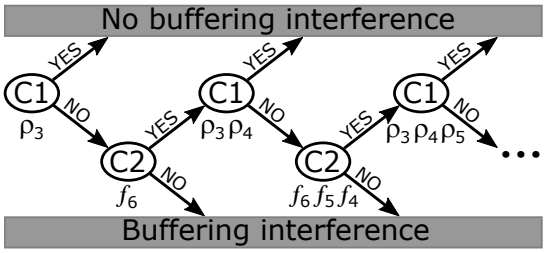


Fig. 9 Buffering interference check for flows f_7 and f_8 from Figure 8

– Otherwise ⇒ buffering interference exists.

Figure 9 demonstrates how Algorithm 1 is applied to the example of flows from Figure 8.

6.3.2 Interference from $\mathcal{F}_D^{DN}$ flows (no additional buffering interference)

We have seen in the previous section that *difd* flows for which Algorithm 1 returns a negative reply cannot cause additional buffering interference. This implies that these flows can be treated as *difu* flows in the worst-case analysis. Therefore, we can formulate the method to compute the WCTT of a flow in the presence of *difo*, *difu* and *difd* flows for which Algorithm 1 returns a negative reply (Theorem 3). Let us first divide *difd* flows into two categories: those for which the buffering interference cannot occur $\mathcal{F}_D^{DN}(f_i)$ and those for which the buffering interference can occur $\mathcal{F}_D^{DB}(f_i)$.

Theorem 3 *Consider the flow-set $\mathcal{F} = \{f_1, f_2, ...f_{n-1}, f_n\}$. Let f_i be the analysed flow, and let $\mathcal{F}_D(f_i) = \mathcal{F}_D^{\emptyset}(f_i) \bigcup \mathcal{F}_D^{U}(f_i) \bigcup \mathcal{F}_D^{D}(f_i)$, i.e., f_i has* difo, difu *and* difd *flows. Moreover, let $\mathcal{F}_D^{D}(f_i) = \mathcal{F}_D^{DN}(f_i)$, i.e. Algorithm 1 produces a negative result for each* difd *flow. The maximum interference that f_i can suffer is at most $I_i^{\emptyset} + I_i^{U} + I_i^{DN}$, where $I_i^{\emptyset}$ can be computed by solving Equation 14, I_i^{U} can be computed by solving Equation 15, and I_i^{DN} can be computed by solving Equation 16.*

$$I_i^{DN} = \sum_{\forall f_j \in \mathcal{F}_D^{DN}(f_i)} \left\lceil \frac{R_i + J_j^R + \overbrace{(R_j - C_j)}^{J_{j\to i}^I} - \gamma_{i,j}^{PRE} - \gamma_{i,j}^{POST}}{T_j} \right\rceil \cdot I_{j\to i} \quad (16)$$

Proof Follows directly from Theorem 2, Algorithm 1 and the discussion in Section 6.3.1. □

6.3.3 Interference from $\mathcal{F}_D^{DB}$ flows (with additional buffering interference)

In this section we focus on *difd* flows for which Algorithm 1 returned a positive reply. These flows cause additional buffering interference to the analysed flow, and they were termed $\mathcal{F}_D^{DB}$ in the previous section. Notice, that flits of $f_j \in \mathcal{F}_D^{DB}(f_i)$ which cause the buffering interference to f_i start grouping in the routers between $\omega(\mathcal{L}_{i,j}^{CD})$ and $\alpha(\mathcal{L}_{i,j}^{CD})$, from back to front (from the former to the latter). Given this observation, let us investigate what is the maximum interference that *already* buffered flits of f_j can cause to f_i (Lemma 6).

Lemma 6 *Consider two flows f_i and f_j. Let $f_j \in \mathcal{F}_D^{DB}(f_i)$. The maximum interference that f_i can suffer from n already buffered flits of f_j cannot exceed the traversal time of those flits through a single link (i.e. $n \cdot d_L$), regardless of their traversal pattern.*

Proof Proven directly. We analyse two scenarios:

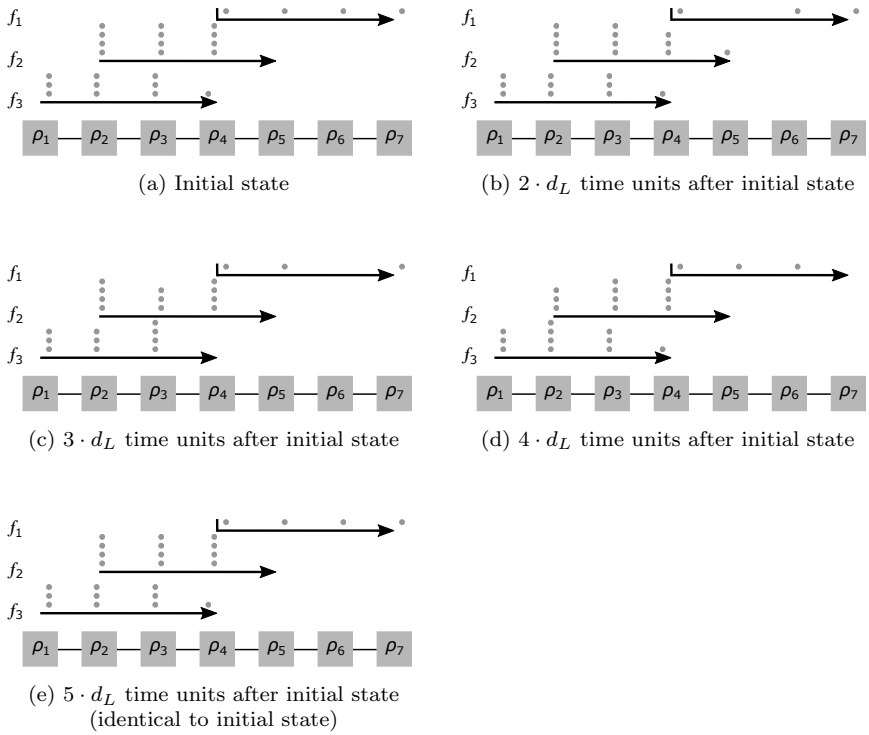
Scenario (i): The n buffered flits leave the contention domain one by one, with enough cycles between departures to stabilise the contention domain (no flits can progress), thus effectively preventing the pipelined departure of buffered flits.

Figure 10 gives a detailed view of this scenario. The highest priority flow f_1 preempts f_2 with 4 flits, and then has a pause for the duration of one d_L due to its upstream interference (not illustrated in Figure 10). Let us assume that this pattern repeats indefinitely. This allows only a single flit of f_2 to depart from $\mathcal{L}_{2,3}^{CD}$, thus effectively preventing buffered flits of f_2 to establish a pipeline. Notice, that after $5 \cdot d_L$ time units the system returns to the initial state. During this time, flow f_3 suffered interference for only one d_L, i.e. its arrival at the destination router has been disrupted only during one link traversal time (at the moment $3 \cdot d_L$ after the initial state there is no flit of f_3 in ρ_4). This corresponds exactly to one flit of f_2 leaving $\mathcal{L}_{2,3}^{CD}$. This process could continue until either of the flows completes its transfer. Regardless, we see that for this scenario the claim of Lemma 6 holds.

Scenario (ii): The n buffered flits leave the contention domain as soon as possible, in a pipelined manner.

Figure 11 gives a detailed view of this scenario. After the existing flits of f_1 leave the network, no new flits of f_1 will appear. This will cause a pipelined transmission of flits of f_2. The full pipeline of f_2 is established $4 \cdot d_L$ time units after the initial state. During that time, 2 flits of f_2 departed from $\mathcal{L}_{2,3}^{CD}$, and that corresponds to the interference which f_3 suffered during that time (its arrival at ρ_4 was interrupted for $2 \cdot d_L$ time units). After that moment, f_2 continues its pipelined transmission, and f_3 can suffer the interference of at most one flit transmission time per each flit of f_2 departing from $\mathcal{L}_{2,3}^{CD}$. Therefore, the claim of Lemma 6 holds for this scenario as well. □

Notice, that even if f_1 would appear again in ρ_4, exactly $3 \cdot d_L$ time cycles after that the system would return back to the initial state, and the interference



(a) Initial state

(b) $2 \cdot d_L$ time units after initial state

(c) $3 \cdot d_L$ time units after initial state

(d) $4 \cdot d_L$ time units after initial state

(e) $5 \cdot d_L$ time units after initial state (identical to initial state)

Fig. 10 Buffered flits of f_2 individually depart from $\mathcal{L}_{2,3}^{CD}$, where $P_1 > P_2 > P_3$ and $\beta = 4$

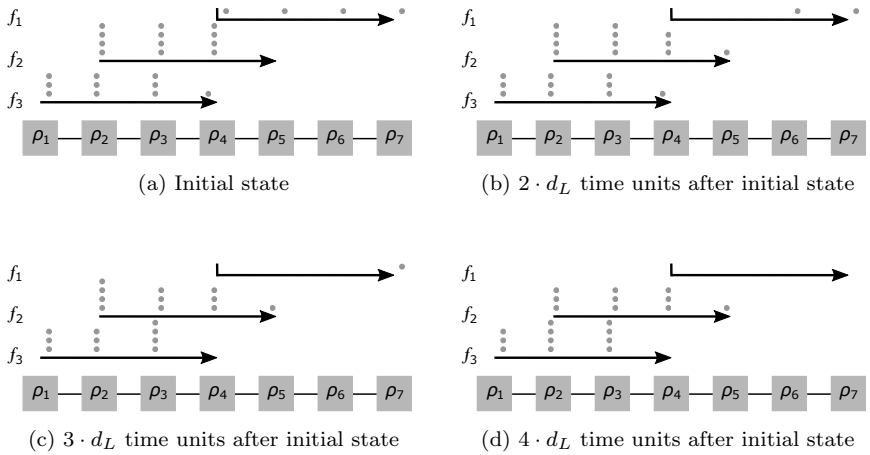
suffered by f_3 during the entire interval of observation would be again equal to the number of flits of f_2 which departed from the contention domain, as proven with Lemma 6.

So far, we have seen that buffered flits grow from the end of the contention domain towards the beginning, and that each buffered flit can cause the interference of at most $1 \cdot d_L$. Now we will discuss the three upper bounds on the amount of buffering interference that f_i suffers from $f_j \in \mathcal{F}_D^{DB}(f_i)$, namely the *size bound* $B_{j\to i}^S$, the *interference bound* $B_{j\to i}^I$ and the *buffer bound* $B_{j\to i}^B$.

Size bound $B_{j\to i}^S$

This bound is developed around the observation that the maximum additional buffering interference that a flow can cause is limited by its size (Theorem 4).

Theorem 4 *Consider two flows f_i and f_j. Let $f_j \in \mathcal{F}_D^{DB}(f_i)$. The maximum buffering interference caused by f_j to f_i has an upper bound which is equal to*



(a) Initial state

(b) $2 \cdot d_L$ time units after initial state

(c) $3 \cdot d_L$ time units after initial state

(d) $4 \cdot d_L$ time units after initial state

Fig. 11 Buffered flits of f_2 depart from $\mathcal{L}_{2,3}^{CD}$ in pipeline, where $P_1 > P_2 > P_3$ and $\beta = 4$

the traversal across a single link of all of its flits, except those that could not be buffered inside the routers within $\mathcal{L}_{i,j}^{CD}$.

Proof Follows straightforwardly from the previous discussion and Lemma 6. The number of potentially buffered flits is equal to the size of the flow, reduced (at least) by what could be buffered in one router. This is because f_j has downstream indirect interference, and hence its flits could be buffered in the router immediately after $\mathcal{L}_{i,j}^{CD}$, without impacting f_i. Thus, the maximum buffering interference has an upper bound equal to $B_{j\to i}^{S}$ (Equation 17).

$$B_{j\to i}^{S} = (\sigma_j - \beta) \cdot d_L \tag{17}$$

□

Interference bound $B_{j\to i}^{I}$

Now we focus on the second bound, which has been identified by Xiong et al (2017). This bound is developed around the observation that buffering interference which f_i suffers from $f_j \in \mathcal{F}_D^{DB}(f_i)$ occurs due to the existence of flow f_k, which causes downstream indirect interference to f_i via f_j. Thus, one flit of f_j is buffered when one flit of f_k progresses. Therefore, the maximum buffering interference of f_j is limited by the maximum interference itself can suffer from f_k, and other flows which are interfering indirectly downstream. We call this bound by *interference*, and it is equal to $B_{j\to i}^{I}$ (Equation 18).

$$B^I_{j\to i} = \sum_{\substack{\forall f_k \in \mathcal{F}_D(f_j) \wedge \\ f_k \notin \mathcal{F}_D(f_i) \wedge \\ order(\alpha(\mathcal{L}^{CD}_{j,k}), \mathcal{L}_j) > \\ order(\omega(\mathcal{L}^{CD}_{i,j}), \mathcal{L}_j)}} \left\lceil \frac{R_j + J^R_k + J^I_{k\to j} - \gamma^{PRE}_{j,k} - \gamma^{POST}_{j,k}}{T_k} \right\rceil \cdot (I_{k\to j} + B_{k\to j}) \tag{18}$$

where

$$B_{k\to j} = \begin{cases} B^{DB}_{k\to j}, & \text{if } f_k \in \mathcal{F}^{DB}_D(f_j) \\ B^{U+DB}_{k\to j}, & \text{otherwise} \end{cases} \tag{19}$$

Notice, that this bound requires to already have computed the buffering interference from f_k to f_j (see the term $B_{k\to j}$ in the above equations). Similar to R_j, for now this term, as well as its individual parts in Equation 19, can be considered as constants, and once we cover all interference scenarios, we will show how to obtain them (Equation 21 and Equation 24).

Buffer bound $B^B_{j\to i}$

Now we propose the third bound, which we call the *buffer* bound. It is developed around the following observation. Since every $f_j \in \mathcal{F}^{DB}_D(f_i)$ has only downstream indirect interference, there exists no upstream interfering flow of f_j which could potentially interrupt the supply of f_j's flits into the contention domain. This is exploited in Theorem 5.

Theorem 5 *Consider two flows f_i and f_j. Let $f_j \in \mathcal{F}^{DB}_D(f_i)$. The maximum buffering interference caused by f_j to f_i has an upper bound which is equal to the traversal across a single link of its flits which could be simultaneously buffered inside the routers within $\mathcal{L}^{CD}_{i,j}$.*

Proof Proven directly. From Lemma 6 we know that each buffered flit can cause additional interference of at most one d_L. We also know that buffering starts at the end of the contention domain and grows towards the beginning. Since f_j does not have upstream indirect interference, it cannot be split into separate sub-packets, but travels continuously, unless preempted by other directly interfering flow, which does not have any effect (as discussed before). While it traverses, the number of buffered flits may vary between the full buffers along the contention domain, when the flow does not progress (see flow f_2 in Figure 10(a) and Figure 11(a)), and all buffers along the contention domain having one flit less than the capacity, when the flow does progress in a pipelined manner (see flow f_2 in Figure 11(d)).

With Lemma 6 it was proven that no additional interference is generated during these fluctuations of the buffer occupancy along the contention domain, but each departed flit causes the interference of exactly one flit traversal. Let

us analyse the time interval between two initial states. Assume that during that time n flits of f_j departed from the contention domain, and hence cause the buffering interference of $n \cdot d_L$ to f_i. However, notice that when the system again reaches the initial state, the departed flits were already replaced by the new ones, which have *not* caused any interference so far. This implies that each of those flits can cause interference for at most one flit traversal. By continuing this reasoning, we see that with several consecutive fluctuations of buffered flits, eventually all flits that could cause interference twice left the contention domain, and the buffers are now occupied with flits that have not caused any interference yet. From this, we can conclude that only the *first* buffered flits which could completely fill the buffers along the contention domain have the possibility to cause interference twice, while any other flit that appears later can cause the interference only once.

Note, that this holds only as long as buffer fluctuations do not cause more substantial changes, the extreme case being an entire emptying/refilling of buffers along the contention domain. For *difd* flows, the buffer fluctuations are limited between the maximum occupancy (see f_2 in Figure 11(a)) and each buffer having one flit less than the maximum (see f_2 in Figure 11(d)). Therefore, the maximum buffering interference caused by f_j to f_i has an upper bound which is equal to the traversal across a single link of flits of f_j which could be simultaneously buffered inside the shared routers within $\mathcal{L}_{i,j}^{CD}$, and it is equal to $B_{j\to i}^{B}$.

$$B_{j\to i}^{B} = \big(|\mathcal{L}_{i,j}^{CD}| - 1\big) \cdot \beta \cdot d_L \tag{20}$$

□

Since all these bounds are safe, we can take the minimum of them:

$$B_{j\to i}^{DB} = min\{B_{j\to i}^{S}, B_{j\to i}^{I}, B_{j\to i}^{B}\} \tag{21}$$

Now, we can compute the interference that the analysed flow suffers from *difo*, *difu* and *difd* flows (Theorem 6).

Theorem 6 *Consider the flow-set $\mathcal{F} = \{f_1, f_2, ...f_{n-1}, f_n\}$. Let f_i be the analysed flow, and let $\mathcal{F}_D(f_i) = \mathcal{F}_D^{\emptyset}(f_i) \bigcup \mathcal{F}_D^{U}(f_i) \bigcup \mathcal{F}_D^{D}(f_i)$, i.e., f_i has* difo, difu *and* difd *flows. The maximum interference that f_i can suffer is at most $I_i^{\emptyset} + I_i^{U} + I_i^{DN} + I_i^{DB}$, where $I_i^{\emptyset}$ can be computed by solving Equation 14, I_i^{U} can be computed by solving Equation 15, I_i^{DN} can be computed by solving Equation 16, and I_i^{DB} can be computed by solving Equation 22.*

$$I_i^{DB} = \sum_{\forall f_j \in \mathcal{F}_D^{DB}(f_i)} \left\lceil \frac{R_i + J_j^R + \overbrace{(R_j - C_j)}^{J_{j\to i}^{I}} - \gamma_{i,j}^{PRE} - \gamma_{i,j}^{POST}}{T_j} \right\rceil \cdot (I_{j\to i} + B_{j\to i}^{DB}) \tag{22}$$

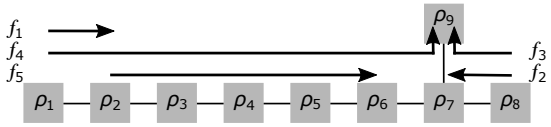


Fig. 12 Example to show that B^B is not safe for *difud* flows ($P_1 > P_2 > P_3 > P_4 > P_5$)

Proof Follows directly from Theorem 4, Theorem 5 and the discussion in Section 6.3.3. □

6.4 Interferece from *difud* $\mathcal{F}_D^{U+D}$ flows

In this section, the focus is on directly interfering flows with both upstream and downstream indirect interference. Similar to *difd* flows, Algorithm 1 can be applied to assess whether the buffering interference can occur. Therefore, let us divide *difud* flows into two groups, those for which the buffering interference cannot occur $\mathcal{F}_D^{U+DN}$ (Algorithm 1 returns false), and those for which the buffering interference can occur $\mathcal{F}_D^{U+DB}$ (Algorithm 1 returns true).

For the former category $\mathcal{F}_D^{U+DN}$, the interference can be computed in a similar way to *difu* (Equation 15) and *difd* (Equation 16) flows which have the same Algorithm 1 result. Equation 23 covers this case.

$$I_i^{U+DN} = \sum_{\forall f_j \in \mathcal{F}_D^{U+DN}(f_i)} \left\lceil \frac{R_i + J_j^R + \overbrace{(R_j - C_j)}^{J_{j\to i}^I} - \gamma_{i,j}^{PRE} - \gamma_{i,j}^{POST}}{T_j} \right\rceil \cdot I_{j\to i} \tag{23}$$

For the latter category $\mathcal{F}_D^{U+DB}$, both the size bound and the interference bound can be applied. However, as indicated in the previous section, the buffer bound cannot be used in this case. This is because upstream indirectly interfering flows can cause severe fluctuations of the buffering along the contention domain, the most drastic one being a complete emptying/refilling of buffers along the contention domain. Therefore, the buffer bound in general case does not apply to flows from $\mathcal{F}_D^{U+DB}$.

Here we provide an illustrative example to demonstrate this point (Figure 12). Flow f_5 suffers interference from flow f_4 which has both upstream (flow f_1) and downstream (flow f_3) indirect interference.

If we compute the buffer bound from f_4 to f_5, assuming $\beta = 5$, we have that $B_{4\to5}^B = |\mathcal{L}_{4,5}^{CD} - 1| \cdot \beta \cdot d_L = 15 \cdot d_L$.

When we consider the flow parameters given in Table 3[1], with $d_R = 0$ and $d_L = 1$, and perform the simulations on a cycle-accurate simulator, we see that the interference which f_4 causes to f_5 exceeds $I_{4\to5} + B_{4\to5}^B$, which is **1015**. In

[1] Other examples with different platform and workload parameters are also possible.

Table 3 Flow parameters for Figure 12

Flow	σ	J_R	D	T
f_1	50	0	100	100
f_2	50	0	100	100
f_3	1000	0	∞	∞
f_4	1000	0	∞	∞
f_5	2000	0	∞	∞

fact, our experiments reported the response time of f_5 of 3130 cycles, which implies that the interference it can suffer from f_4 is **1125**. This is because the existence of f_1, f_2 and f_3 periodically causes a complete buffer filling and emptying of flow f_4 along $\mathcal{L}_{4,5}^{CD}$, and as discussed before, in scenarios with such fluctuations of buffered flits, the buffer bound is not safe.

Therefore, the buffering interference from *difud* flows can be computed as follows:

$$B_{j\to i}^{U+DB} = min\{B_{j\to i}^{S}, B_{j\to i}^{I}\} \tag{24}$$

And finally, the maximum interference that a flow f_i can suffer from all its interfering flows is covered with Theorem 7.

Theorem 7 *Consider the flow-set $\mathcal{F} = \{f_1, f_2, ...f_{n-1}, f_n\}$. Let f_i be the analysed flow. The maximum interference that f_i can suffer is at most $I_i^{\emptyset} + I_i^{U} + I_i^{DN} + I_i^{DB} + I_i^{U+DN} + I_i^{U+DB}$, where $I_i^{\emptyset}$ can be computed by solving Equation 14, I_i^{U} can be computed by solving Equation 15, I_i^{DN} can be computed by solving Equation 16, I_i^{DB} can be computed by solving Equation 22, I_i^{U+DN} can be computed by solving Equation 23, and I_i^{U+DB} can be computed by solving Equation 25.*

$$I_i^{U+DB} = \sum_{\forall f_j \in \mathcal{F}_D^{U+DB}(f_i)} \left\lceil \frac{R_i + J_j^R + \overbrace{(R_j - C_j)}^{J_{j\to i}^{I}} - \gamma_{i,j}^{PRE} - \gamma_{i,j}^{POST}}{T_j} \right\rceil \cdot (I_{j\to i} + B_{j\to i}^{U+DB}) \tag{25}$$

Proof Follows directly from the aforementioned discussion. □

After defining all interference types, now we can describe the computation of the interference term of Condition 2 for the absence of buffering interference. Recall, the objective is to compute the maximum interference that a preempting flow may suffer, and test whether that interference is sufficient to cause buffering interference to the analysed flow. Let f_i be the analysed flow, f_j be the preempting flow for which we test the existence of buffering interference, and f_k be the higher-priority flow whose contention domain with f_j starts on the link p (e.g. flows f_8, f_7, f_6 and link between ρ_3 and ρ_4 in

Figure 8). Moreover, let p be the currently analysed link in Algorithm 1. The term $inf(f_k, f_j, p)$ can be obtained as follows:

$$inf(f_k, f_j, p) = \begin{cases} \text{Contribution of } f_k \text{ to } I_j^{\emptyset} \text{ (from Equation 14)} & \text{if } f_k \in \mathcal{F}_D^{\emptyset}(f_j^p) \\ \text{Contribution of } f_k \text{ to } I_j^{U} \text{ (from Equation 15)} & \text{if } f_k \in \mathcal{F}_D^{U}(f_j^p) \\ \text{Contribution of } f_k \text{ to } I_j^{DN} \text{ (from Equation 16)} & \text{if } f_k \in \mathcal{F}_D^{DN}(f_j^p) \\ \text{Contribution of } f_k \text{ to } I_j^{DB} \text{ (from Equation 22)} & \text{if } f_k \in \mathcal{F}_D^{DB}(f_j^p) \\ \text{Contribution of } f_k \text{ to } I_j^{U+DN} \text{ (from Equation 23)} & \text{if } f_k \in \mathcal{F}_D^{U+DN}(f_j^p) \\ \text{Contribution of } f_k \text{ to } I_j^{U+DB} \text{ (from Equation 25)} & \text{if } f_k \in \mathcal{F}_D^{U+DB}(f_j^p) \end{cases} \tag{26}$$

Note, that in Equation 26, f_j should be treated as if it would terminate at the router after p, hence the term f_j^p. The only exception is that the total interference of f_k to f_j is computed assuming the interval of interest corresponding to the traversal of f_j across its entire path R_j.

Finally, the WCTT of flow f_i (Equation 27) can be obtained by summing up its traversal delay and the interference it suffers.

$$R_i = \overbrace{C_i}^{\substack{traversal \\ delay}} + \overbrace{I_i^{\emptyset}}^{\substack{difo \\ interference}} + \overbrace{I_i^{U}}^{\substack{difu \\ interference}} + \overbrace{I_i^{DN} + I_i^{DB}}^{\substack{difd \\ interference}} + \overbrace{I_i^{U+DN} + I_i^{U+DB}}^{\substack{difud \\ interference}} \tag{27}$$

7 Experimental Evaluation

In this section, we conduct an experimental evaluation of the proposed method. First, we compare it against the following state-of-the-art techniques: the method of Xiong et al (2017) (referred to as $SOTA$), and the method of Indrusiak et al (2018) (referred to as $SOTA^+$). The comparison is performed for various platform and workload configurations, while the aspects of interest are schedulability guarantees (**Experiment 1**), WCTT bounds and runtime complexities (**Experiment 2**). Then, we investigate how different VC buffer sizes affect schedulability guarantees (**Experiment 3**). After that, we assess the efficiency of the proposed approach by comparing the obtained WCTT bounds against corresponding values observed via simulations for synthetic workloads (**Experiment 4**) and for a use-case of an autonomous driving vehicle application (**Experiment 5**). Finally, we conclude the experimental evaluation by analysing the hardware requirements of the proposed approach (**Experiment 6**).

7.1 Experimental Setup

The analysis and simulation parameters are summarised in Table 4. An asterisk sign denotes a randomly generated value, assuming a uniform distribution.

Table 4 Analysis and simulation parameters

NoC topology	**2-D mesh**
NoC size	**8 × 8**
Routing method	**X-Y routing**
Router frequency	**2GHz**
Routing latency + link traversal latency	**3 + 1 cycles**
Link width	**4B**
Flit size	**4B**
Flow source router coordinates	⟨ **[1 - 8]*, [1 - 8]*** ⟩
Flow destination router coordinates	⟨ **[1 - 8]*, [1 - 8]*** ⟩
Flow size	**[1 - 128]* KB**
Flow priority assignment policy	**Rate monotonic**
Flow deadline = Flow period	**[0.01 - 1]*** ***msec***
Simulated time	**1** ***sec***
Simulator	**SPARTS (extended) Nikolić et al (2011)**
Hardware	**Intel i5-2520M, with 4GB of RAM**

Note, that if during the creation of a flow its source and destination routers have the same coordinates, the destination router coordinates are generated again.

7.2 Experiment 1: Schedulability Guarantees

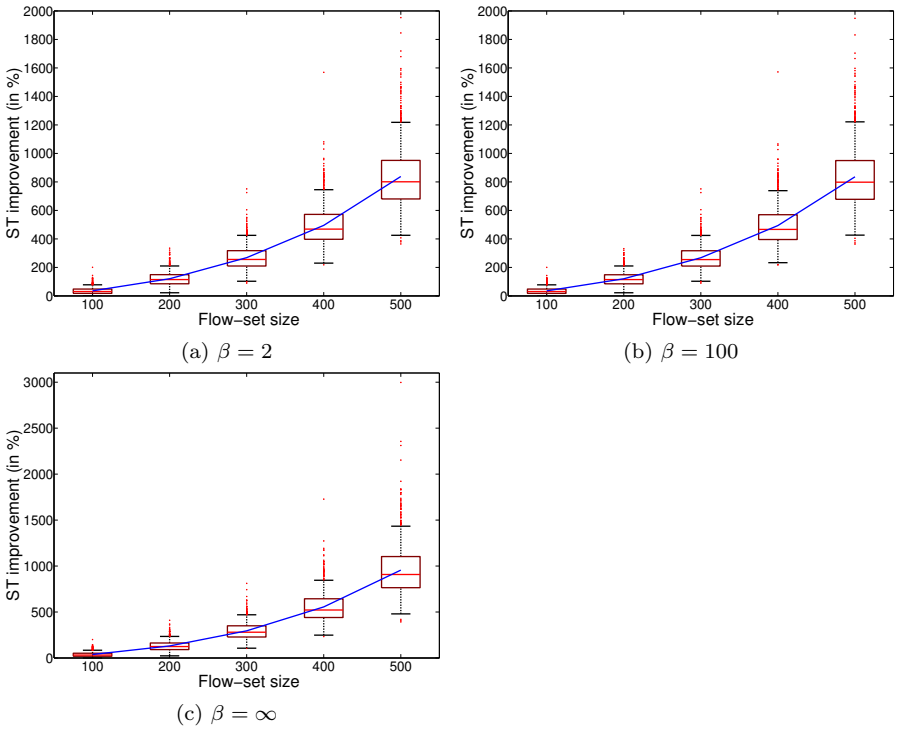
In this experiment, we perform a comparison of the proposed approach against both $SOTA$ and $SOTA^+$. The aspects of interest are schedulability guarantees. The comparison is conducted in the form of a *sensitivity analysis*. Assuming a certain configuration with platform θ and workload $\mathcal{F}$, the schedulability test is performed. If a flow-set is schedulable, the sizes of all flows are uniformly increased, and the test is performed again. Similarly, if a flow-set is unschedulable, the sizes of all flows are uniformly decreased, and the test is performed again. This process is repeated until a threshold value is found (called the *schedulability threshold* ST), where a flow-set is schedulable, however, any increase in sizes of flows would render it unschedulable. Of course, the bigger the ST value is, the more efficient the method is.

Let ST_{sota} be the schedulability threshold value obtained for $SOTA$, and let ST_{new} be the schedulability threshhold value obtained for the proposed approach. Then, the following metric is used to assess the improvements of the proposed method over $SOTA$:

$$imp = \frac{ST_{new} - ST_{sota}}{ST_{sota}} \cdot 100\%$$

Similarly, let ST_{sota^+} be the schedulability threshold value obtained for $SOTA^+$. Then, the following metric is used to asses the improvements of the proposed method over $SOTA^+$.

$$imp = \frac{ST_{new} - ST_{sota^+}}{ST_{sota^+}} \cdot 100\%$$



(a) $\beta = 2$ (b) $\beta = 100$

(c) $\beta = \infty$

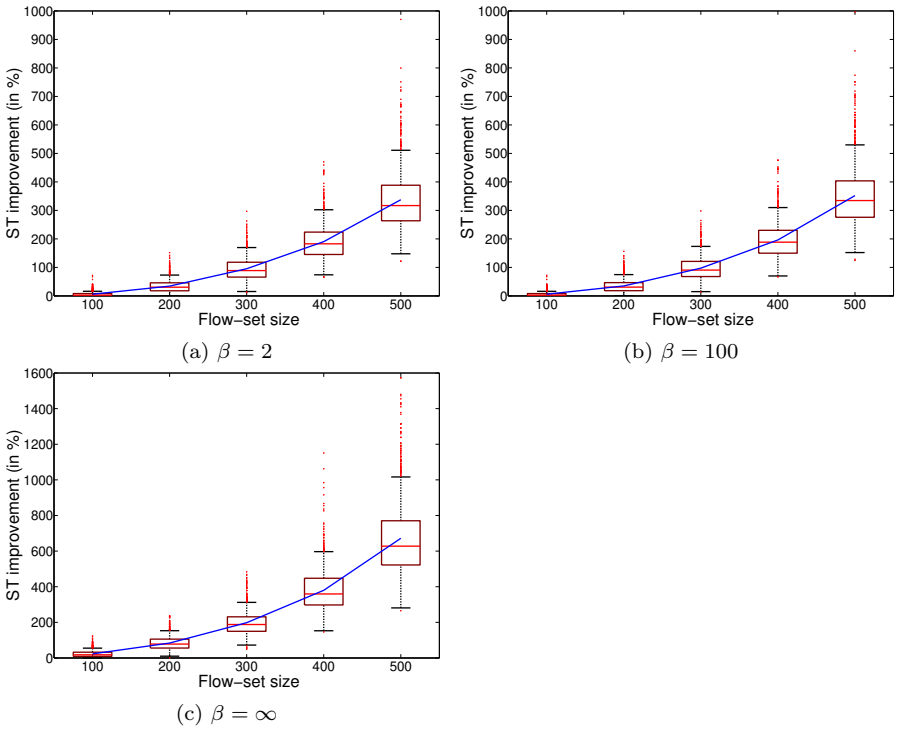
Fig. 13 Experiment 1: ST improvements over $SOTA$ (method of Xiong et al (2017))

Since the proposed approach dominates both methods (i.e. always produces the same or bigger ST values), the improvements have only positive values.

We repeated the aforementioned comparison for varying platform and workload configurations. Namely, we used 3 different values for buffer sizes of VCs (recall the symbol β in Table 1): (i) each buffer can store only 2 flits, (ii) each buffer can store at most 100 flits, and (iii) each buffer can store an entire packet (buffer size set to the maximum packet size). Additionally, we varied the flow-set size, and observed the improvements for flow-sets with 100, 200, 300, 400 and 500 flows. For each of these unique configurations we have generated 1000 flow-sets and computed with the above metrics the improvements over both $SOTA$ and $SOTA^+$. The results are illustrated in Figure 13[2] and Figure 14, respectively.

Figure 13 demonstrates that the improvements curve is always convex, and that the gains grow with the increasing flow-set size. This is expected, because more flows lead to more complex contention scenarios, which the proposed method handles efficiently, while the state-of-the-art method significantly over-

[2] In Figures 13-21, the box-edges represent the 25^{th} percentile (q_1) and the 75^{th} 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 as an outlier. Additionally, the blue lines connect the mean values of the respective categories.



(a) $\beta = 2$ (b) $\beta = 100$

(c) $\beta = \infty$

Fig. 14 Experiment 1: ST improvements over $SOTA^+$ (method of Indrusiak et al (2018))

approximates. It is also visible that the improvements grow with growing β. This can be explained with the fact that for larger values of β the proposed approach efficiently identifies scenarios where buffering interference does not occur, while the state-of-the-art method unconditionally considers it. Thus, the biggest improvements are observed for 500 flows and $\beta = \infty$, where the proposed method, on average, allows to accommodate the workload which is 9 times bigger than the one which could be accommodated by $SOTA$. The biggest observed improvement ratio for an individual flow-set is slightly less than 30 (3000% in Figure 13(c)).

Figure 14 shows the improvements against $SOTA^+$. The trends are very similar to the previous case, in a sense that bigger flow-set sizes and bigger β both contribute to more significant improvements. The improvements against $SOTA^+$ are on average 40% smaller than the improvements against $SOTA$, which can be attributed to a more efficient treatment of flows with only downstream indirect interference (see the difference between Equation 6 and Equation 7 in Section 5). Nonetheless, the improvements of our method against $SOTA^+$ are still substantial. Again, the best results are observed for 500 flows and $\beta = \infty$ where the proposed method, on average, allows to accommodate the workload which is 6 times bigger than the one which could be

accommodated by $SOTA^+$. The biggest observed improvement ratio for an individual flow-set is 15.75 (1575% in Figure 14(c)).

7.3 Experiment 2: WCTT Improvements and Scalability

In this experiment, we evaluate the improvements of the proposed method against both $SOTA$ and $SOTA^+$ with respect to WCTTs of individual flows. Assuming a given flow-set, first we obtained the ST for $SOTA$. Then, we adjusted the sizes of all flows by the obtained ST. This was done to make sure that the tested flow-sets will indeed be schedulable with both the proposed method and the state-of-the-art methods used for comparison. After that, we derived and compared the WCTTs of all flows in the following way. Let $WCTT_{sota}$ be the WCTT of one flow obtained with $SOTA$, and let $WCTT_{new}$ be the WCTT of the same flow obtained with the proposed method. Then, the following metric is used to describe the improvement of the new method over $SOTA$:

$$imp = \frac{WCTT_{sota} - WCTT_{new}}{WCTT_{sota}} \cdot 100\%$$

This process was repeated for all flows of the flow-set.

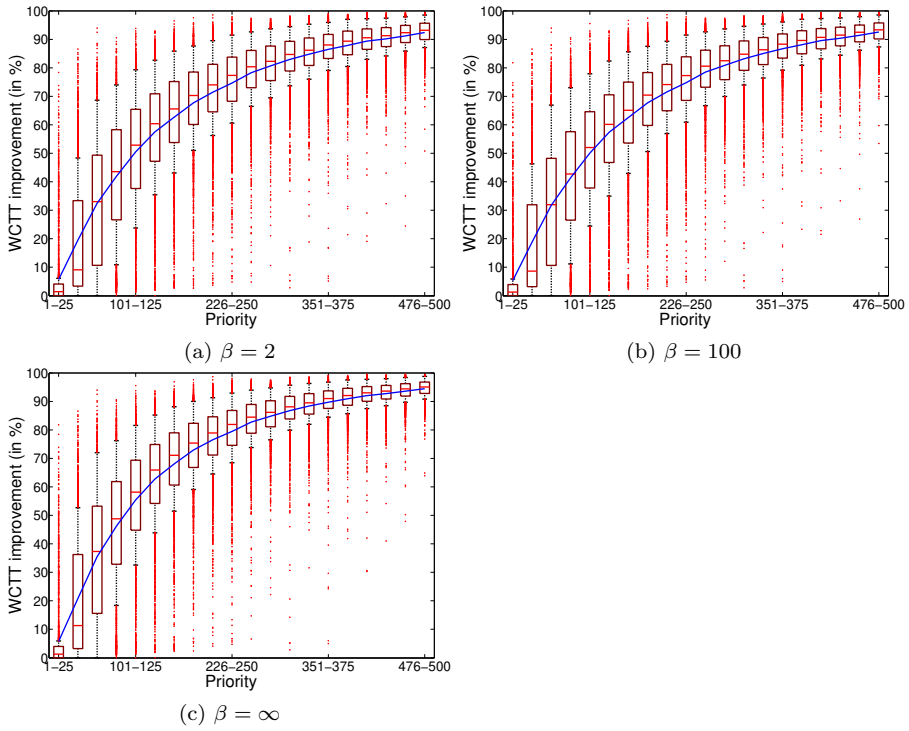
Similarly, we repeat the aforementioned procedure for $SOTA^+$. The following metric is used to describe the improvement of the new method over $SOTA^+$:

$$imp = \frac{WCTT_{sota^+} - WCTT_{new}}{WCTT_{sota^+}} \cdot 100\%$$

We repeated the experiment for 1000 flow-sets, each with 500 flows. For better visualisation, the results were organised in priority groups, e.g. $[1 - 25], [26 - 50], ..., [476 - 500]$ (a smaller value denotes a higher priority). We also repeated the same experiment for different values of β. The results of a comparison against $SOTA$ are illustrated in Figure 15, while the results of a comparison against $SOTA^+$ are illustrated in Figure 16.

From Figure 15 we conclude that as flow priorities decrease, the improvements become more apparent. The gains are the biggest for the lowest priority flows, and for all 3 tested values of β they asymptotically converge towards 100%. This is expected, because lower-priority flows suffer more interference and contention scenarios are more complex, which the proposed approach efficiently handles. It is visible that the improvement curve is concave across the entire domain for all 3 tested configurations of β. As in the previous experiment, bigger values of β yield more improvements, and the best results are achieved for $\beta = \infty$ (Figure 15(c)). This coincides with the finding of Experiment 1.

Figure 16 demonstrates the improvements of the proposed method against $SOTA^+$. The conclusions are similar to the previous case, the gains grow with decreasing priorities. It is also visible that for $\beta = 2$ (Figure 16(a)) and



(a) $\beta = 2$ (b) $\beta = 100$

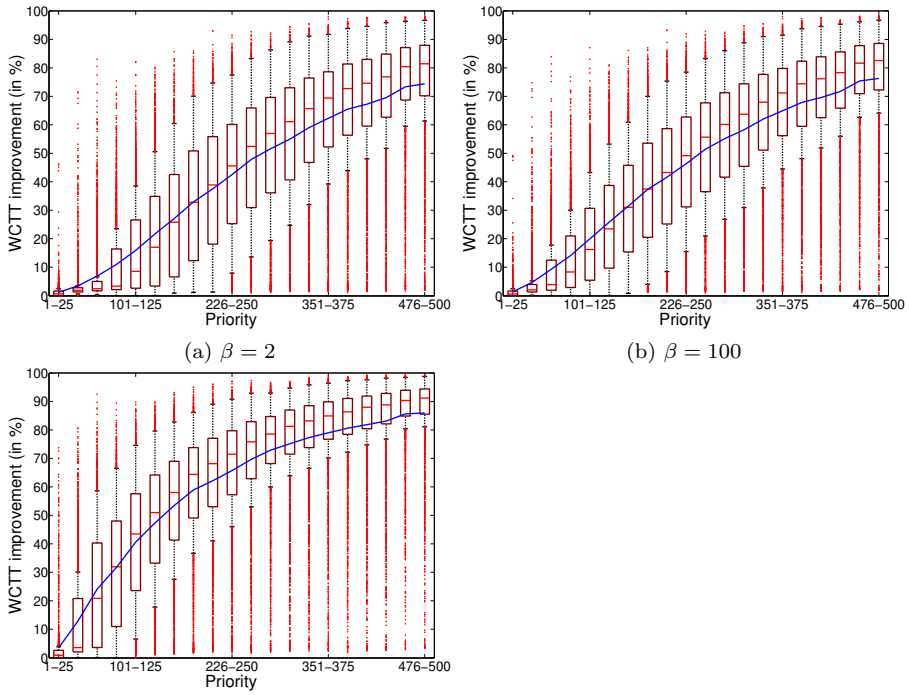
(c) $\beta = \infty$

Fig. 15 Experiment 2: WCTT improvements over $SOTA$ (method of Xiong et al (2017))

$\beta = 100$ (Figure 16(b)) the improvements curve is slightly convex for higher priorities, and slightly concave for lower priorities, with the inflection point near the middle of the domain (priorities around 250). Conversely, for $\beta = \infty$ (Figure 16(c)), the improvement curve is concave across the entire domain. Again, it is visible that bigger values of β yield more improvements.

Finally, for all evaluated methods we recorded execution times, so as to assess their runtime complexities and discuss their scalability potentials. The results are illustrated in Figure 17, where the distribution of execution times for the proposed method is illustrated. Moreover, mean values for all three approaches are also illustrated.

From Figure 17 it is visible that for $\beta = 2$ and $\beta = 100$, the proposed method takes longer time to compute WCTTs. This is expected, due to the fact that the proposed approach is indeed computationally more complex. However, the time penalty is not significant at all. This can be explained with the fact that the proposed method produces substantially tighter results than $SOTA$ and $SOTA^+$, and consequently, requires less iterations to converge to WCTT values. This effect is especially emphasised for $\beta = \infty$, where the proposed method, despite its higher complexity, computes WCTTs faster than $SOTA$ and $SOTA^+$.



(a) $\beta = 2$ (b) $\beta = 100$

(c) $\beta = \infty$

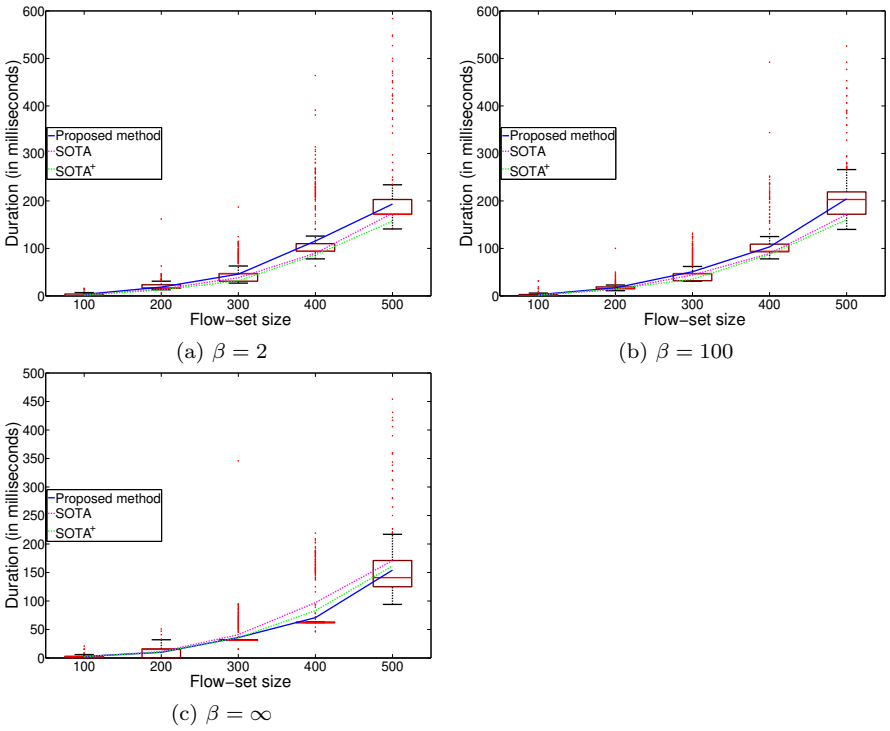
Fig. 16 Experiment 2: WCTT improvements over $SOTA^{+}$ (method of Indrusiak et al (2018))

Moreover, even for flow-sets with 500 flows, the proposed method derives WCTT values, on average, in 200 milliseconds, while the maximum observed computation time is slightly less than 600 milliseconds. This implies that the proposed method is indeed scalable and applicable to workloads consisting of hundreds of flows.

7.4 Experiment 3: Effect of Buffer Sizes on Schedulability Guarantees

The objective of this experiment is to assess the effects of VC buffer sizes on derived schedulability guarantees. Assuming a given flow-set, we varied the number β in the following range $[2, 10, 100, 1000, 10000, \infty]$, and observed how ST values change. First, let ST_x be a ST value obtained for β from the following range $\{2, 10, 100, 1000, 10000\}$. Similarly, let ST_∞ be a corresponding value for $\beta = \infty$. Then, the following metric is used to assess the decrease (penalty) in the derived schedulability guarantees of the approach with limited β, against the one with $\beta = \infty$ (used as a baseline):

$$pen = \frac{ST_x}{ST_\infty} \cdot 100\%$$



(a) $\beta = 2$ (b) $\beta = 100$

(c) $\beta = \infty$

Fig. 17 Experiment 2: Scalability and runtime complexity of the proposed approach, $SOTA$ (method of Xiong et al (2017)) and $SOTA^{+}$ (method of Indrusiak et al (2018))

The experiment was performed for 1000 flow-sets, each with 500 flows. The results are illustrated in Figure 18. It is visible that, on average, the negative effect of using smaller VC buffers is around 11%. With an increase in the buffer sizes, counter-intuitively, the obtained STs drop. One explanation might be that any increase in buffer sizes renders the buffer bound less applicable. At the same time, the increase in buffer sizes is not so significant to nullify the buffering interference. Hence, the derived STs slightly drop. This interesting finding and the corresponding explanation will be revisited in Experiment 4. It is also visible in Figure 18 that after a certain threshold (in our experiment it is $\beta = 10000$), the buffer sizes were such that almost all buffering interference could be avoided, and hence we see a significant increase in derived STs.

The results suggest that, if it is not possible to provide a platform with VC buffer sizes which allow to avoid buffering interference in majority of cases, it is more efficient to use a less resourceful platform with only $\beta = 2$. Please note, that in our experiment the threshold for performance jump was so high ($\beta = 10000$) because we designed our experiment in such a way to test the limits of the proposed method, and hence loaded the NoC with the maximum load which it could sustain while still guaranteeing the schedulability. This approach caused all flow sizes to be significantly inflated during the search for

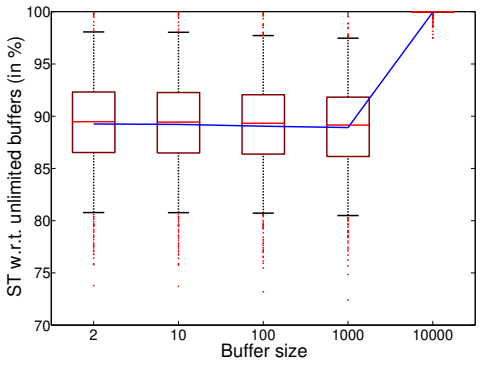


Fig. 18 Experiment 3: STs for varied buffer sizes, relative to STs for unlimited buffers

ST, and hence $\beta = 10000$ was the threshold point. Also note, that in realistic scenarios flows can be significantly smaller, and hence the threshold point would be reached for smaller values of β.

7.5 Experiment 4: Method Efficiency

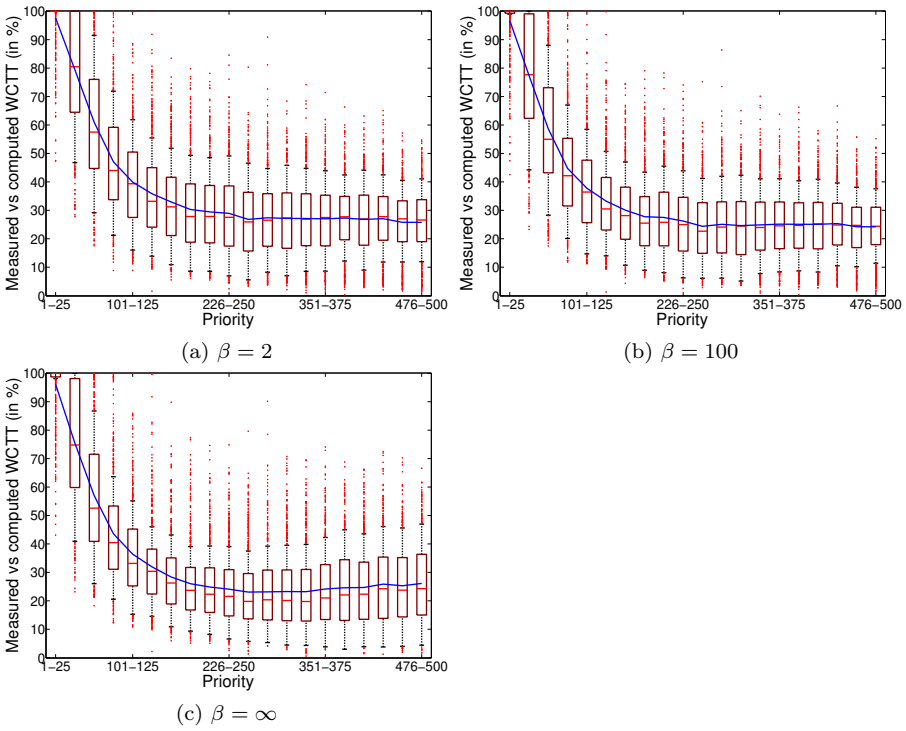
The focus of this experiment is on estimating the efficiency of the proposed method with respect to derived WCTTs. To do so, we implemented a cycle-accurate simulator of the platform described in Section 3.1, by extending the simulator SPARTS (Nikolić et al (2011)). We assessed the tightness of derived WCTT bounds by comparing them against the corresponding WCTT values observed during simulations.

The experiment was conducted as follows. First, we used the same flow-sets and ST values from Experiment 2, and corrected flow sizes accordingly. Then, we simulated the execution of 1 second, which, on average, took 5 hours per flow-set. After the simulation was completed, we collected the observed WCTTs of all flows. Then we compared them with the WCTT bounds obtained by the proposed method in Experiment 2. Let $WCTT_{sim}$ be the observed $WCTT$ of one flow, and let $WCTT_{new}$ be the WCTT bound obtained with the proposed method. The tightness of the derived bound is expressed with the following metric:

$$tightness = \frac{WCTT_{sim}}{WCTT_{new}} \cdot 100\%$$

Due to the fact that simulations take much longer to finish, we collected the results for 20 flow-sets. Again, for better visualisation, the results were organised in priority groups, e.g. $[1-25], [26-50], ..., [476-500]$. We repeated the experiment for different values of β. The results are illustrated in Figure 19.

From Figure 19 it is visible that the tightness of bounds decreases with decreasing priorities, which is an expected result, because complex contention scenarios which are associated with the lower-priority flows are less likely to



(a) $\beta = 2$ (b) $\beta = 100$

(c) $\beta = \infty$

Fig. 19 Experiment 4: Observed WCTTs (via simulation), relative to analytical WCTTs

be captured during a limited simulation time. The decrease in tightness is exponential and asymptotically converges to 25%, however, with the longer simulation time this value could be improved. Therefore, a more extensive experimental evaluation is a potential future activity. Moreover, it is visible that the results are very similar for different values of β, which implies that the method scales with respect to VC buffer sizes, and is equally applicable to platforms with small buffers (e.g. $\beta = 2$) and huge buffers (e.g $\beta = \infty$).

If we combine the findings of Experiment 3 (increasing buffer sizes may have a negative effect on derived guarantees) with the findings of this experiment (tightness scales with β), we can conclude that even the simulation results for larger values of β are *worse* than the corresponding ones for smaller values of β. This clarifies that the phenomenon observed in Experiment 3 is not a property of the proposed analysis method, but in fact the inherent characteristic of priority-preemptive NoCs. This finding supports the conclusions from Experiment 3, and it is of crucial importance for system designers, because it suggests that there are two viable strategies: (i) use platforms with small buffer sizes (e.g. $\beta = 2$), or (ii) use platforms with sufficiently large buffers which allow to (almost) completely mitigate the effects of the buffering interference (e.g. $\beta = 10000$ in our experiments). Any intermediate solutions

would be more expensive than the former one (more hardware resources), and at the same time would provide worse results.

7.6 Experiment 5: Use-Case of Autonomous Driving Vehicle Application

In this experiment, we also assess the efficiency of the proposed method. We do it in the same way as in Experiment 4, by comparing analytically obtained WCTT values against the corresponding ones observed via simulations. But instead of using a synthetic workload, the workload is modelled after a use-case of an autonomous driving vehicle application (Shi et al (2010)). The use-case consists of 33 functionalities producing totally 38 traffic flows. For a more detailed description of the use-case, a reader is advised to consult the work of Shi et al (2010).

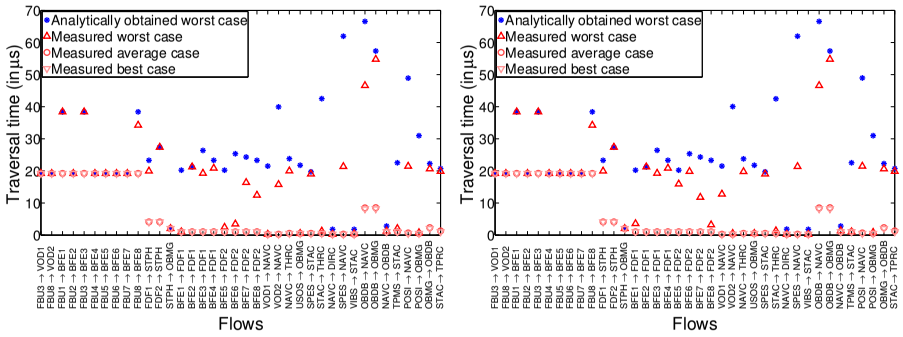
The experiment was conducted in the following way. First, WCTT bounds were obtained by the proposed method for $\beta = 2$. Then, we simulated the execution of 100 seconds, and for each flow we collected the following values: (i) the observed worst-case traversal time, (ii) the observed average-case traversal time, and (iii) the observed best-case traversal time. Then, the same process was repeated for $\beta = 100$ and $\beta = \infty$. The values of interest are plotted in Figure 20.

From Figure 20 it is visible that in majority of cases, the average and the best case are almost identical. This implies that flows usually traverse without any contentions. However, in scenarios where contentions do occur, the traversal times are significantly inflated. From Figure 20 we can also observe that in most cases the proposed method derives tight WCTT bounds (a small gap between the analytically obtained and the corresponding measured worst-case), and that trend is evident for all configurations of β. The remark from the previous experiment is also valid here; the simulations were performed for only a limited amount of time (100 seconds of simulated time), and longer simulation runs would even further reduce the aforementioned gap.

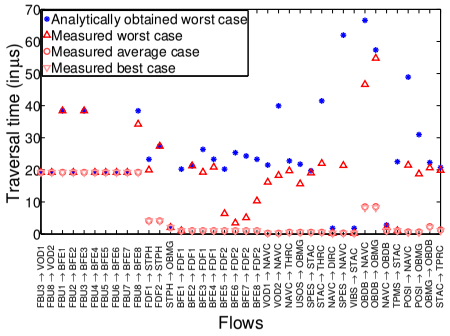
We also computed the maximum number of port contentions (a necessary requirement to guarantee a dedicated per-port VC to each flow), and found out that the workload from this use-case can be accommodated by a platform with only 4 VCs.

7.7 Experiment 6: Hardware Requirements

In this experiment, we assess the hardware requirements of the proposed model. Recall, that the number of available VCs within the platform should be at least equal to the maximum number of contentions for any port, which is a requirement that guarantees a dedicated per-port VC to each flow (Nikolić et al (2013)). We used the flow-sets from Experiment 1 (randomly generated sources and destinations) and computed the number of needed VCs for each of them. This process was repeated for varying flow-set size. The results are illustrated in Figure 21.



(a) $\beta = 2$ (b) $\beta = 100$



(c) $\beta = \infty$

Fig. 20 Experiment 5: Traversal times for the autonomous driving vehicle application (analytical worst case, measured worst case, measured average case and measured best case)

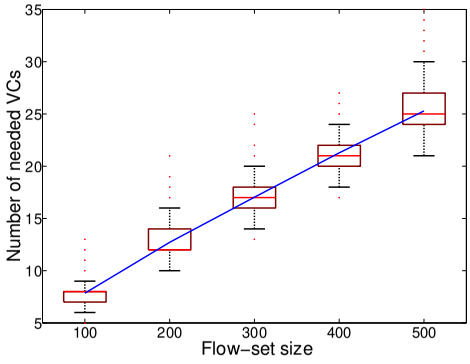


Fig. 21 Experiment 6: Hardware requirements

From Figure 21 it is visible that the number of needed VCs scales linearly with the increasing number of flows in the flow-set. This is expected, because a 2-D mesh is a scalable NoC topology. Additionally, we see that, even for the massive workloads of 500 flows, on average, only 25 VCs are needed. Please note, that this result is based on randomly generated traffic sources and des-

tinations, assuming the X-Y routing policy. It has already been demonstrated that with a thoughtful mapping (Nikolić et al (2013)) and a thoughtful routing (Nikolić and Pinho (2017)) this number can be significantly reduced (on average, by 25% and by 40%, respectively). Given that there already exist platforms with 8 VCs (e.g. Intel (2010)), we can expect that the forthcoming generations of real-time oriented many-cores with priority-preemptive NoCs will have a dozen or more VCs. With thoughtful mapping and routing, such platforms could successfully accommodate workloads comprised of several hundreds of traffic flows.

8 Conclusions and Future Work

In this work, we proposed a novel method for the worst-case analysis of traversal times of network traffic flows, deployed upon a priority-preemptive NoC. Compared to the state-of-the-art techniques, our approach renders more flow-sets schedulable, and also yields substantially tighter upper-bounds on the worst-case traversal times. By employing the proposed method, resource over-provisioning can be mitigated to a large extent, and significant design-cost reductions can be achieved. Moreover, we implemented a cycle-accurate simulator of the assumed NoC architecture, and used it to assess the tightness of derived WCTT bounds. Finally, we reached an interesting conclusion that larger virtual channel buffers do not necessarily lead to better results, and in many cases can be counter-productive, which is a very important finding for system designers.

As a future work, we plan to extensively evaluate the proposed approach with additional use-cases and benchmarks. Also, extending the method, so as to make it applicable to flow-sets with arbitrary deadlines and platforms with fewer virtual channels is a promising future work activity. Finally, how to (i) map flows to cores, (ii) assign priorities to flows and (iii) assign paths to flows are relevant problems which remain to be addressed.

References

Benini L, De Micheli G (2002) Networks on chips: a new soc paradigm. The Computer Journal 35(1):70 –78

Burns A, Harbin J, Indrusiak L (2014) A wormhole noc protocol for mixed criticality systems. In: Proceedings of the 35th IEEE Real-Time Systems Symposium

Dally W (1992) Virtual-channel flow control. IEEE Transactions on Parallel and Distributed Systems 3(2):194 –205

Dally W, Seitz C (1987) Deadlock-free message routing in multiprocessor interconnection networks. IEEE Transactions on Computers

Dasari D, Nikolić B, Nelis V, Petters SM (2013) Noc contention analysis using a branch and prune algorithm. ACM Transactions on Embedded Computing Systems

Diemer J, Ernst R (2010) Back suction: Service guarantees for latency-sensitive on-chip networks. In: International Symposium on Networks-on-Chip

de Dinechin BD, van Amstel D, Poulhiès M, Lager G (2014a) Time-critical computing on a single-chip massively parallel processor. In: Proceedings of the 17th Conference on Design Automation and Test in Europe

de Dinechin BD, Durand Y, van Amstel D, Ghiti A (2014b) Guaranteed services of the noc of a manycore processor. In: Proceedings of the International Workshop on Network on Chip Architectures

Ferrandiz T, Frances F, Fraboul C (2011) A network calculus model for spacewire networks. In: Proceedings of the 17th IEEE Conference on Embedded and Real-Time Computing and Applications

Goossens K, Dielissen J, Radulescu A (2005) Aethereal network on chip: concepts, architectures, and implementations. IEEE Design & Test of Computers

Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis - the symta/s approach. IEE Proceedings - Computers and Digital Techniques 152(2)

Hu J, Marculescu R (2003) Energy-aware mapping for tile-based noc architectures under performance constraints. In: Proceedings of the 8th Asia and South Pacific Design Automation Conference

Indrusiak LS (2014) End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. Journal of System Architecture

Indrusiak LS, Harbin J, Burns A (2015) Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In: Proceedings of the 27th Euromicro Conference on Real-Time Systems

Indrusiak LS, Burns A, Nikolić B (2016) Analysis of buffering effects on hard real-time priority-preemptive wormhole networks. Technical report arxiv:1606.02942

Indrusiak LS, Burns A, Nikolić B (2018) Buffer-aware bounds to multi-point progressive blocking in priority-preemptive nocs. In: Proceedings of the 21st Conference on Design Automation and Test in Europe

Intel (2010) Single-Chip-Cloud Computer.
`www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf`

Intel (2013) Intel® Xeon Phi™.
`http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html`

Kalray (2014) MPPA-256 Manycore Processor.
`www.kalrayinc.com/kalray/products/#processors`

Kasapaki E, Schoeberl M, Sørensen RB, Müller C, Goossens K, Sparsø J (2016) Argo: A real-time network-on-chip architecture with an efficient gals implementation. IEEE Transactions on Very Large Scale Integration Systems

Kashif H, Patel H (2014) Bounding buffer space requirements for real-time priority-aware networks. In: Proceedings of the 19th Asia and South Pacific Design Automation Conference

Kashif H, Patel H (2016) Buffer space allocation for real-time priority-aware networks. In: Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium

Kashif H, Gholamian S, Patel H (2014) Sla: A stage-level latency analysis for real-time communication in a pipelined resource model. IEEE Transactions on Computers 99

Kavaldjiev NK, Smit GJM (2003) A survey of efficient on-chip communications for soc. In: Proceedings of the 4th Symposium on Embedded Systems

Liu M, Becker M, Behnam M, Nolte T (2015a) Improved priority assignment for real-time communications in on-chip networks. In: Proceedings of the 23rd International Conference on Real-Time Networks and Systems

Liu M, Behnam M, Nolte T (2015b) A stochastic response time analysis for communications in on-chip networks. In: Proceedings of the 21st IEEE Conference on Embedded and Real-Time Computing and Applications

Liu M, Becker M, Behnam M, Nolte T (2016a) Scheduling real-time packets with non-preemptive regions on priority-based nocs. In: Proceedings of the 22nd IEEE Conference on Embedded and Real-Time Computing and Applications

Liu M, Becker M, Behnam M, Nolte T (2016b) Tighter time analysis for real-time traffic in on-chip networks with shared priorities. In: International Symposium on Networks-on-Chip

Liu M, Becker M, Behnam M, Nolte T (2017) A tighter recursive calculus to compute the worst case traversal time of real-time traffic over nocs. In: Proceedings of the 22nd Asia and South Pacific Design Automation Conference

Mesidis P, Indrusiak L (2011) Genetic mapping of hard real-time applications onto noc-based mpsocs – a first approach. In: 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip

Millberg M, Nilsson E, Thid R, Jantsch A (2004) Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In: Proceedings of the 7th Conference on Design Automation and Test in Europe, vol 2, pp 890–895 Vol.2

Ni LM, McKinley PK (1993) A survey of wormhole routing techniques in direct networks. The Computer Journal 26

Nikolić B, Petters SM (2014a) Edf as an arbitration policy for wormhole-switched priority-preemptive nocs – myth or fact? In: Proceedings of the 14th International Conference on Embedded Software

Nikolić B, Petters SM (2014b) Real-time application mapping for many-cores using a limited migrative model. Real-Time Systems Journal

Nikolić B, Pinho LM (2017) Optimal minimal routing and priority assignment for priority-preemptive real-time nocs. Real-Time Systems Journal

Nikolić B, Awan MA, Petters SM (2011) SPARTS: Simulator for power aware and real-time systems. In: Proceedings of the 8th IEEEInternational Conference on Embedded Software and Systems

Nikolić B, Ali HI, Petters SM, Pinho LM (2013) Are virtual channels the bottleneck of priority-aware wormhole-switched noc-based many-cores? In: Proceedings of the 21st International Conference on Real-Time Networks

and Systems
Nikolić B, Yomsi PM, Petters SM (2014) Worst-case communication delay analysis for many-cores using a limited migrative model. In: Proceedings of the 20th IEEE Conference on Embedded and Real-Time Computing and Applications
Nikolić B, Indrusiak LS, Petters SM (2016a) A tighter real-time communication analysis for wormhole-switched priority-preemptive nocs. Technical report arxiv:1605.07888
Nikolić B, Pinho LM, Indrusiak LS (2016b) On routing flexibility of wormhole-switched priority-preemptive nocs. In: Proceedings of the 22nd IEEE Conference on Embedded and Real-Time Computing and Applications
Paukovits C, Kopetz H (2008) Concepts of switching in the time-triggered network-on-chip. In: Proceedings of the 14th IEEE Conference on Embedded and Real-Time Computing and Applications, pp 120–129
Racu A, Indrusiak L (2012) Using genetic algorithms to map hard real-time on noc-based systems. In: 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip
Rambo EA, Ernst R (2015) Worst-case communication time analysis of networks-on-chip with shared virtual channels. In: Proceedings of the 18th Conference on Design Automation and Test in Europe
Sayuti M, Indrusiak L (2013) Real-time low-power task mapping in networks-on-chip. In: IEEE Computer Society Annual Symposium on VLSI
Schoeberl M (2007) A time-triggered network-on-chip. In: Proceedings of the 17th International Conference on Field-Programmable Logic and Applications
Schoeberl M, Abbaspour S, Akesson B, Audsley N, Capasso R, Garside J, Goossens K, Goossens S, Hansen S, Heckmann R, Hepp S, Huber B, Jordan A, Kasapaki E, Knoop J, Li Y, Prokesch D, Puffitsch W, Puschner P, Rocha A, Silva C, Sparsø J, Tocchi A (2015) T-crest: Time-predictable multi-core architecture for embedded systems. Journal of System Architecture
Shi Z, Burns A (2008a) Priority assignment for real-time wormhole communication in on-chip networks. In: Proceedings of the 29th IEEE Real-Time Systems Symposium
Shi Z, Burns A (2008b) Real-time communication analysis for on-chip networks with wormhole switching. In: International Symposium on Networks-on-Chip
Shi Z, Burns A (2010) Schedulability analysis and task mapping for real-time on-chip communication. Real-Time Systems Journal
Shi Z, Burns A, Indrusiak LS (2010) Schedulability analysis for real time on-chip communication with wormhole switching. International Journal on Embedded and Real-Time Communication Systems
Song H, Kwon B, Yoon H (1997) Throttle and preempt: a new flow control for real-time communications in wormhole networks. In: Proceedings of the 1997 International Conference on Parallel Processing
Stefan RA, Molnos A, Goossens K (2012) daelite: A tdm noc supporting qos, multicast, and fast connection set-up. IEEE Transactions on Computers

63(3)

Tilera (2012) TILE64™ Processor. www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf

Tobuschat S, Ernst R (2017) Real-time communication analysis for networks-on-chip with backpressure. In: Proceedings of the 20th Conference on Design Automation and Test in Europe

Xiong Q, Lu Z, Wu F, Xie C (2016) Real-time analysis for wormhole noc: Revisited and revised. In: Proceedings of the 26th ACM Great Lakes Symposium on VLSI

Xiong Q, Wu F, Lu Z, Xie C (2017) Extending real-time analysis for wormhole nocs. IEEE Transactions on Computers 66(9)