CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Worst-Case Communication Delay Analysis for Many-Cores using a Limited Migrative Model

Borislav Nikolic

Patrick Meumeu Yomsi

Stefan M. Petters

CISTER-TR-140202

Version:

Date: 2/6/2014

Worst-Case Communication Delay Analysis for Many-Cores using a Limited Migrative Model

Borislav Nikolic, Patrick Meumeu Yomsi, Stefan M. Petters

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: borni@isep.ipp.pt, pamyo@isep.ipp.pt, smp@isep.ipp.pt

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

Abstract

A steady increase in the number of cores within many-core platforms causes increasing contentions for the interconnect medium and leads to non-negligible latencies of intercore communication. In order to study the worst-case execution times of applications, it is no longer sufficient to only take into account schedulability requirements, but the communication delays also have to be considered. In this paper we focus on the worst-case communication delays of applications, assuming a Limited Migrative Model (LMM). LMM is an approach based on the multi-kernel paradigm - a promising step towards scalable and predictable many-cores. The contribution of this paper is threefold. First, we extend LMM by allowing inter-application communication, and adapt the existing worst-case communication delay analysis, to make it applicable to the enhanced model. Second, we propose a novel analysis. Finally, we compare these two methods. The experiments show that the new approach renders tighter upper-bound estimates in more than 90% of the cases, while demonstrating a comparable runtime performance.

Worst-Case Communication Delay Analysis for Many-Cores using a Limited Migrative Model

Borislav Nikolić, Patrick Meumeu Yomsi and Stefan M. Petters
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto, Portugal
Email: {borni, pamyo, smp}@isep.ipp.pt

***Abstract*—A steady increase in the number of cores within many-core platforms causes increasing contentions for the interconnect medium and leads to non-negligible latencies of inter-core communication. In order to study the worst-case execution times of applications, it is no longer sufficient to only take into account schedulability requirements, but the communication delays also have to be considered. In this paper we focus on the worst-case communication delays of applications, assuming a *Limited Migrative Model* (*LMM*). *LMM* is an approach based on the *multi-kernel* paradigm - a promising step towards scalable and predictable many-cores. The contribution of this paper is threefold. First, we extend *LMM* by allowing inter-application communication, and adapt the existing worst-case communication delay analysis, to make it applicable to the enhanced model. Second, we propose a novel analysis. Finally, we compare these two methods. The experiments show that the new approach renders tighter upper-bound estimates in more than 90% of the cases, while demonstrating a comparable runtime performance.**

I. Introduction

The miniaturisation process in the semiconductor technology reached the stage where further processing power enhancements related to single-core systems are no longer affordable [1]. In order to satisfy the ever increasing demand for more powerful computational devices, chip manufacturers took a design paradigm shift [2], [3] and started interconnecting multiple cores within a single chip. Nowadays, platforms containing several cores (*multi-cores*) and more than a dozen of cores (*many-cores*) have become commonplace in many scientific areas, most notably *high performance computing*, while are an emerging technology in others, like *real-time embedded systems*. This paper focuses on the latter.

In many-cores, the *Network-on-Chip* (NoC [4]) architecture has become the predominant interconnect medium due to its scalability potential [5]. On such platforms the *wormhole switching* technique [6] is widely applied (e.g. [2], [3]) because of its good throughput and small buffering requirements [5].

In order to apply many-cores in the real-time embedded domain, an OS paradigm is needed, such that it allows the system designer to exploit the full potential of the underlying hardware, and yet assures predictability and scalability [7], [8], which are essential prerequisites for deriving execution guarantees. *Limited Migrative Model* (LMM) [9] is an approach based on the foundations of the *multi-kernel paradigm* [7], which is a novel OS design (e.g. [7], [10], [11]), and a promising step towards scalable and predictable many-cores.

Contribution: This paper focuses on the worst-case communication delays of applications, residing within a NoC-based, wormhole-switched many-core platform using LMM. Specifically, our aim is to derive, for each application, a tight upper-bound estimate on the duration of all communication it performs within its minimum inter-arrival period. First, we enhance LMM to include inter-application communication, and adapt the existing analysis [9], so as to make it applicable to the extended model. Second, we present a novel analysis. Finally, we compare these two approaches. The experiments show that the new method provides tighter upper-bounds and also demonstrates a comparable runtime performance.

II. LMM Motivation and Concepts

Unpredictability, scalability, the need for load balancing and core shutdowns are some of the key issues that make the integration of many-cores into the real-time embedded domain a challenging subject. The existing state-of-the-art methodologies are not efficient in addressing all of the aforementioned challenges, and this significantly limits their scope of application, as highlighted below.

All approaches can be broadly classified into two categories: *Non-Migrative Approaches* and *Migrative Approaches*. Non-migrative approaches [12] are in the scheduling theory also known as *Fully-Partitioned Approaches*. Each application is migrationless, and, at design time, is statically assigned to a core where it has to execute. Migrative approaches are further divided into *Semi-Partitioned Approaches* and *Global Approaches*. The former category [13], [14] considers a static assignment of an application to a core (or a subset of cores if it migrates). A migrative application also follows design time decisions, i.e. always executes the prescribed fraction of work on each of assigned cores. Oppositely, global approaches [15] assume that each application may execute on any core.

As is evident, fully partitioned and semi-partitioned approaches are inflexible, and can be very inefficient in scenarios with substantial load changes, where runtime load balancing and/or core shutdowns are required for energy/thermal management, performance enhancements or fault tolerance reasons. Conversely, global approaches inherently support load balancing, however, due to the necessity to maintain global structures (e.g. ready-queue) in a centralised entity, scalability issues arise [7], [8], and serious challenges occur when attempting real implementations [16].

LMM [9] is a recently introduced approach in the real-time embedded domain. Its novelty is twofold. First, it poses a constraint that each application may execute *only* on a subset

of cores, which are decided at design time. During runtime, an application may migrate between the candidate cores. Second, the release/migration decisions of each application are made by the application itself, which removes the requirement of a mandatory centralised scheduling entity. This contributes the scalability, and yet gives the possibility to perform runtime load balancing. So far, LMM appears to be a promising approach, and this work is motivated by that reasoning. More details about LMM are given in Section IV-B.

III. Related Work

Contrary to the popular belief, the wormhole switching technique [6] is not a novelty neither in academia, nor in industry. However, it has been largely neglected because an alternative *store-and-forward switching* was providing satisfactory results [5]. But as the amount of transferred data kept increasing, the buffering within routers became a challenge [5]. This brought wormhole switching back into focus, and some of present many-cores employ this technique, e.g. [2], [3].

When organising the access to the interconnect medium, platforms employ different techniques, which can be classified as either contentionless or contention aware. An example of the former category is the Aethereal platform [17] where a time-division-multiple-access method is used to organise the access to the interconnect resources. If contentions are allowed and a platform provides only a single virtual channel, complex traffic interference patterns may occur [18]. Several analyses were proposed to obtain upper-bounds on the worst-case delays [19]–[22], however, due to the complex interference patterns, the obtained values may be overly pessimistic [19].

Conversely, if multiple virtual channels are employed [23], [24] two benefits arise: (i) by avoiding idle routers the throughput significantly increases [23], [24] and (ii) traffic preemptions can be implemented [25]. Shi and Burns [26] proposed the worst-case delay analysis, assuming per-priority virtual channels, flit-level preemptions and per-packet distinctive priorities.

The aforementioned approaches are based on the assumption that traffic routes are known at design time. Yet, LMM causes non-deterministic packet routes (Section VI-A), thus rendering these approaches not suitable to our model. Nikolić and Petters [9] recognised this problem and proposed a naive and simplistic method to compute the worst-case communication delays. The limitations of this method are: (i) it considers only intra-application traffic and (ii) it treats path non-determinism in a pessimistic way, by assuming that each application may suffer interference from all higher priority traffic that exists within the network, irrespective of (im)possible contentions. In this paper we tackle the traffic non-determinism via new techniques, and propose a novel worst-case communication delay analysis. Moreover, for LMM are available a schedulability analysis [27] and the worst-case memory traffic analysis [28].

IV. Model

A. Platform

We consider a contention-aware NoC-based many-core platform, comprised of $m \times n$ tiles interconnected with a 2D mesh, where each tile contains a single core and a single router, e.g. Tilera family of processors [3]. The platform employs a static, dimension-ordered XY routing mechanism, which is deadlock and livelock free [29]. With this policy, packets firstly travel on the x-axis, and upon reaching the x-coordinate of the destination, traverse along the y-axis. Moreover, data transfer is implemented via a wormhole switching technique. This means that, prior to sending, each data packet is divided into small elements of fixed size called *flits*. The first flit establishes the path, and the rest follows in a pipeline manner. Additionally, the target platform provides virtual channels, which are implemented as additional flit-sized buffers within each port of each router. A virtual channel may be used to store one flit of a preempted traffic packet. A prerequisite is that the number of virtual channels is at least equal to the maximum number of contentions at any router. This requirement assures that each packet has an available virtual channel within each port along its path, which is a realistic assumption [30]. Indeed, on a 10×10 grid, the workload of 400 packets would require on average only 8 virtual channels [30]. In this paper, terms *packet* and *message* are used interchangeably.

B. Software Layers

LMM builds on top of a multi-kernel paradigm, meaning that each core runs an independent kernel instance. Kernels communicate among each other and constitute the basic communication infrastructure. Each kernel exposes some of its functionalities to applications located on its core via system calls. In order to interact with other applications residing on the same or other cores, applications invoke available system calls. Each kernel performs a local scheduling on its core.

An application is restricted to execute only on a subset of cores, which are decided at design time. On each of the selected cores the execution code of that application exists, constituting an entity called *dispatcher*. Dispatchers of the same application (each located on a different core) communicate and decide whether the migration will occur, and if so, which core (dispatcher) is the destination. The aforementioned communication is called *agreement protocol* [9], and its purpose is to elect one dispatcher (*master dispatcher*), which will subsequently release on its core the next job on behalf of the entire application. A released job has to complete on the master's core, it can not migrate (no job-level migrations). Applications are single-threaded, implemented as recurrent tasks, therefore at any time instant there can be only one master per application.

Upon completing the execution, a master is responsible for initiating a new instance of the agreement protocol. Other dispatchers that participate in the communication are called *slave dispatchers*. If the outcome of the protocol is that the migration occurs, the previous master becomes the slave, while the newly elected dispatcher becomes the master. Additionally, the execution context has to be transferred from the old to the new master. The protocol execution is termed *intra-application communication*. Perceived from the application's perspective, its dispatchers exchange one master token, thus a master is only a temporary role of a dispatcher. We call this property *master volatility* and it has several implications which are covered in Section VI-A. Figure 1 gives a graphical representation

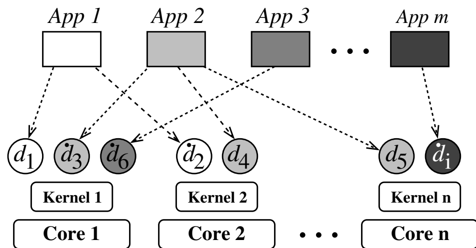

Fig. 1: Limited migrative model - LMM

of LMM, where current master dispatchers can be distinguished by dots over their names. Notice, that in LMM each core performs a local scheduling (similar to fully partitioned approaches), while each application still has a possibility to migrate (similar to global approaches). These benefits come at a price of protocol communication, and the more migrative flexibility the applications have (the number of dispatchers), the more substantial are the communication delays (see Figure 2). The purpose of this example is to help the reader perceive the notion of LMM and its scalability potential. The simulation parameters are given in Table I (Section VII).

Note, the existing definition of LMM and the accompanying analysis [9] only consider intra-application traffic, i.e. agreement protocols. In this work we extend LMM and allow applications to communicate with each other, e.g. for synchronisation or data-sharing purposes. We term this process *inter-application communication*, and it is performed by an exchange of messages between current master dispatchers of interacting applications. We also extend the existing analysis [9] (Section VI-B), so as to match the enhanced model.

The execution workload is described by an application-set $\mathcal{A} \stackrel{\text{def}}{=} \{a_1, a_2, ..., a_{m-1}, a_m\}$. Each application $a\langle C_a, T_a, P_a, \mathcal{D}_a, \mathcal{M}_a\rangle$ is characterised by its execution time C_a, a minimum inter-arrival period T_a, a unique priority P_a, a set of dispatchers $\mathcal{D}_a$ and a set of messages $\mathcal{M}_a = \mathcal{M}_I(a) \cup \mathcal{M}_S(a) \cup \mathcal{M}_R(a)$, where $\mathcal{M}_I(a)$ is a set of intra-application messages, while $\mathcal{M}_S(a)$ and $\mathcal{M}_R(a)$ represent sets of sent and received inter-application messages, respectively. Every message m_{ij} from the source dispatcher d_i to the destination dispatcher d_j is characterised by (i) a priority it inherits from the application of the sender dispatcher d_i, (ii) the amount of the transferred content – $size(m_{ij})$ and (iii) a path – $path(m_{ij})$, consisting of $nhops(m_{ij})$ hops. If both d_i and d_j belong to the same application a, then m_{ij} is an intra-application message, i.e. $m_{ij} \in \mathcal{M}_I(a)$. Otherwise, if $d_i \in \mathcal{D}_a$ and $d_j \in \mathcal{D}_{a'}$, it is an inter-application message, i.e. $m_{ij} \in \mathcal{M}_S(a) \wedge m_{ij} \in \mathcal{M}_R(a')$.

V. Background and Preliminaries

Before we present the main contribution of this work, let us introduce several basic properties of the NoC contention analysis, which are important for understanding our approach.

Prop. 1: An isolation delay of each message is equal to the latency of its first flit to reach the destination, augmented by the processing time of all flits at the destination router (Equation 1). l_{sw} is the latency of the arbitration and the crossbar switching, l_t is the latency to transfer one flit between successive routers, while $size(f)$ represents the size of the flit.

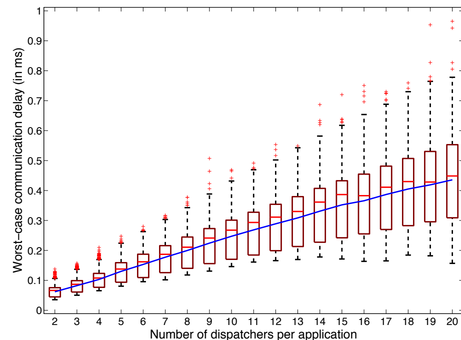

Fig. 2: The impact of the number of dispatchers on the communication delay

$$l(m) = nhops(m) \times (l_{sw} + l_t) + \left\lceil \frac{size(m)}{size(f)} \right\rceil \times l_t \quad (1)$$

Prop. 2: Due to flit-level preemptions, a message can be blocked by the lower-priority traffic, at most for the duration of one flit traversal, within each router on its path (Equation 2).

$$b(m) = nhops(m) \times (l_{sw} + l_t) \quad (2)$$

Prop. 3: A message m of an application a can be preempted only by higher priority messages which share a part of the path with it, called *directly interfering messages*. Let $\mathcal{M}_D(m)$ be a set of directly interfering messages of m. Formally:

$$\forall m' \in \mathcal{M}_{a'} : (P_{a'} > P_a \wedge path(m') \cap path(m) \neq \emptyset) \Rightarrow m' \in \mathcal{M}_D(m) \quad (3)$$

Prop. 4: Interference caused by a single preemption of any $m' \in \mathcal{M}_D(m)$ to m is equal to the sum of its isolation and blocking delays – $l(m') + b(m')$.

Note, in scenarios with a single virtual channel, a message might suffer interference from other higher priority messages which do not directly interfere with it, i.e. do not share any part of the path with it [18], [26]. However, when assuming per-priority virtual channels (as in this work), such messages can not cause interference, but can influence the occurrence patterns of directly interfering messages [26], [28]. Thus, assuming periodic occurrences of directly interfering messages is an unsafe assumption, and in the analysis we have to consider their worst-case occurrence patterns. This is a well-known issue in the wormhole switching and a detailed explanation can be found in the following works [26], [28].

VI. Proposed Approach

A. Challenges of LMM

Due to the master volatility in LMM (i.e. the ability of every dispatcher to become a master), it is not possible to predict at design time which dispatchers will be elected masters of their applications at a particular time instant at runtime. Furthermore, as each master initiates the agreement protocol and performs the inter-application communication on behalf of the entire application, message paths will highly depend on which dispatcher performs that role. The leftmost part of Figure 3 shows agreement protocol messages of one application, captured at two different time instants. Depending on which dispatcher is the current master (emphasized circle), messages may traverse completely different paths. Similar problem occurs when analysing the inter-application traffic. As every dispatcher of both interacting applications can be a master, a single inter-application message may take any of

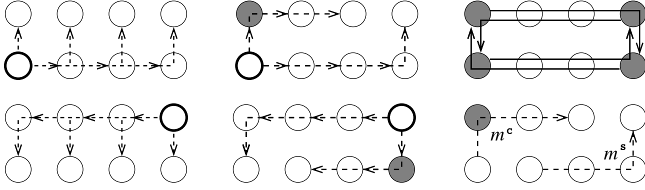

Fig. 3: Intra-Application Communication

the routes given in the left part of Figure 4. As is evident, non-deterministic message paths render traditional worst-case communication delay analyses inapplicable to LMM.

B. Extension of the Existing Approach [9]

As already discussed, Nikolić and Petters [9] proposed the analysis for LMM, assuming only intra-application traffic. Their approach is path-abstracting and circumvents the problem of non-deterministic traffic routes. Specifically, the analysis exploits the fact that isolation and blocking delays of a message (Equations 1-2) are the consequence of 2 properties: (i) its size – $size(m)$ and (ii) its path length – $nhops(m)$. The size of the message is deterministic, but the path is not. Let $maxhops(a)$ be the maximum distance between any two dispatchers of one application. As $maxhops(a)$ is an upper-bound on the path length of each intra-application message m, the analysis covers the worst-case by assuming that each m traverses that distance, i.e. $nhops(m) = maxhops(a)$. Furthermore, it is assumed that any higher priority message (i) will also traverse its longest possible path and (ii) will cause interference, irrespective of its potential route. Subsequently, this concept was applied to the entire traffic.

In this section we extend that reasoning and apply the same approach to inter-application traffic. Let a and a' be two interacting applications with the maximum distance between the furthest dispatchers equal to $maxhops(a, a')$. Consequently, each inter-application message m between a and a' will traverse *at most* that distance, so $nhops(m) = maxhops(a, a')$ covers the worst-case. Similarly, the conservative assumption is that any higher priority inter-application message (i) will traverse its longest possible path and (ii) will cause interference.

The isolation and blocking delays of an application, $del_I(a)$ and $del_B(a)$ are equal to the sum of isolation and blocking delays of its intra-application messages and sent inter-application messages, respectively (Equations 4-5).

$$del_I(a) = \sum_{\forall m \in \mathcal{M}_I(a) \cup \mathcal{M}_S(a)} l(m) \quad (4) \qquad del_B(a) = \sum_{\forall m \in \mathcal{M}_I(a) \cup \mathcal{M}_S(a)} b(m) \quad (5)$$

In order to consider the worst-case interference, we have to compute the maximum traffic that every higher priority application a' can generate within the observed time interval. For that, we use Theorem 1, taken from the existing analysis [9].

Theorem 1. *[9] The number of protocol executions of any application a within the time interval t can be at most $1 + \left\lceil \frac{t - C_a}{T_a} \right\rceil$.*

Since for each application the protocol is performed once per its inter-arrival period, the result of Theorem 1 also presents the maximum number of inter-arrivals of a given application.

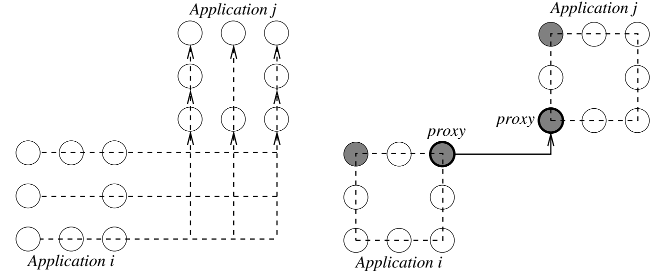

Fig. 4: Inter-Application Communication

Therefore, the maximum interference suffered from all higher priority applications within one inter-arrival period T_a, termed $del_N(a)$, can be computed by multiplying the maximum number of their inter-arrivals with the maximum traffic they can generate per one inter-arrival period (Equation 6).

$$del_N(a) = \sum_{\forall a' \in \mathcal{A}: P_{a'} > P_a} \left(1 + \left\lceil \frac{T_a - C_{a'}}{T_{a'}} \right\rceil\right) \times (del_I(a') + del_B(a')) \quad (6)$$

Note, a message can be stalled by other same-priority messages of the same application with which it shares the same virtual channel. A safe assumption is that all messages of the same application traverse sequentially and hence each of them contributes to the isolation delay of the application with its entire traversal latency (see Equation 4). The same reasoning will be used in the novel approach (Section VI-C). Now, the worst-case communication delay of an application is equal to the sum of these terms: $del(a) = del_I(a) + del_B(a) + del_N(a)$.

C. Novel approach

Main idea: Recent insights into priority-based, wormhole-switched NoCs have proved that the dominant factor in the worst-case delay of a message is not the length of its path, but rather the interference it suffers [30]. Thus, the pessimism related to the approach presented in the previous section can be attributed mostly to the conservative assumption that all higher priority traffic can cause interference, irrespective of (im)possible contentions. Motivated by this reasoning, we propose a novel approach, which relies on enforcing constraints, in order to make LMM traffic deterministic and predictable.

Constraint 1. *Dispatchers of an application can be positioned only on the edges of a rectangular $x \times y$ structure, such that no corner is left unoccupied and $x, y \in \mathbb{N}$. The special case is a line-like shape, where one or both dimensions of the shape are equal to* "1".

Constraint 2. *Intra-application messages travel only on the edges of the shape its application is forming, and re-routing occurs where needed to comply with the global XY routing policy. An individual message rotation (i.e. clockwise or counterclockwise) is chosen such that the traversal distance is minimised.*

The middle part of Figure 3 illustrates how dispatchers should be placed and messages consequently routed. Shaded dispatchers denote locations where reroutings occur. Reroutings are fast on-core routines, performed in an interrupt-like manner which can be, for example, implemented by instrumenting the Hardwall™ technology of Tilera platforms [3].

1) **Supermessages and Proxies:** Constraints 1-2 make deterministic the part of the network that intra-application traffic uses. To make the traffic master independent as well, we introduce the following concepts: *supermessages* and *proxies*.

Definition 1 (Supermessage). *A supermessage is a message which connects (i) diagonally-placed dispatchers if an application has a rectangular shape, or (ii) terminal dispatchers if an application has a line-like shape.*

An application with a rectangular shape has 4 supermessages (see the top right part of Figure 3), 2 with clockwise orientation ($\widehat{m_{cw1}}$, $\widehat{m_{cw2}}$), and 2 with counter-clockwise orientation ($\widehat{m_{cc1}}$, $\widehat{m_{cc2}}$). A line-like shape yields 2 supermessages – $\widehat{m_{l1}}$, $\widehat{m_{l2}}$, and does not involve reroutings. Due to space limitations, in the rest of this work only rectangular shapes are analysed. Indeed, any conclusion reached for a rectangular shape can be applied to a line-like shape by considering only one supermessage of each orientation and treating the other one as non-existent (i.e. $\widehat{m_{l1}} = \widehat{m_{cw1}}; \widehat{m_{l2}} = \widehat{m_{cc1}}; \widehat{m_{cw2}} = \emptyset; \widehat{m_{cc2}} = \emptyset;$).

Theorem 2. *Any intra-application message of an application can be expressed by at most 2 distinctive, same-orientation supermessages, with at most* 1 *rerouting.*

Proof. Proven by contradiction. An intra-application message m assumes the orientation such that the distance between the dispatchers is minimised (Constraint 2). If σ is the circumference of an application shape, it holds that $nhops(m) \leq \frac{\sigma}{2}$. As each supermessage $\widehat{m}$ connects diagonal corners of a shape, for every $\widehat{m}$ it holds that $nhops(\widehat{m}) = \frac{\sigma}{2}$.

Assume that m can be expressed with at least 3 same-orientation supermessages. Note, as there are only two same-orientation supermessages (e.g. $\widehat{m_1}$ and $\widehat{m_2}$), hence one of them has to appear twice in m, i.e. the sequence would be either $\{\widehat{m_1}, \widehat{m_2}, \widehat{m_1}\}$, or $\{\widehat{m_2}, \widehat{m_1}, \widehat{m_2}\}$. In either case, the middle supermessage entirely belongs to m, while the first and the last belong with fractions $\epsilon_1 > 0$ and $\epsilon_2 > 0$, respectively. Hence:

$$nhops(m) = \epsilon_1 + nhops(\widehat{m_1}) + \epsilon_2 = \epsilon_1 + nhops(\widehat{m_2}) + \epsilon_2 = \epsilon_1 + \frac{\sigma}{2} + \epsilon_2 > \frac{\sigma}{2}$$

A contradiction has been reached. Additionally, as reroutings occur only on places where supermessages meet, and since any message can be expressed by at most 2 distinctive supermessages, it can involve at most 1 rerouting. □

Notice that supermessages are master-independent and their number is significantly smaller than the number of possible message paths. Thus, the intuitive idea behind our novel approach is to transform every intra-application message into the corresponding supermessage(s) with eventual rerouting, and perform the analysis on such a model.

Now we apply similar approach to inter-application traffic via $proxy$ dispatcher.

Definition 2 (Proxy). *A proxy dispatcher is a dispatcher which is selected at design time, and which participates in the inter-application communication. It mediates the communication between its master and the proxy dispatcher of the interacting application.*

An illustrative example of Definition 2 is given in the right part of Figure 4. In this scenario an inter-application message is divided into 5 different components: 1) a message from the master sender to its proxy, 2) a rerouting on the core of the proxy sender, 3) a message between the proxies, 4) a rerouting on the core of the proxy receiver, 5) a message from the proxy receiver to its master. Proxies are decided at design time, thus a message m_{ij} between proxy dispatchers d_i and d_j is also deterministic and master-independent. Note: An application can have multiple proxies, each responsible for the communication with a different application.

Theorem 3. *Any inter-application message can be expressed by (i) at most 2 distinctive, same-orientation supermessages on the sender's side, (ii) a message between a proxy sender and a proxy receiver, (iii) at most 2 distinctive, same orientation supermessages on the receiver's side and (iv) at most 4 reroutings.*

Proof. Proven directly. A message between a master sender and its proxy complies with the rules of the intra-application traffic, hence, according to Theorem 2, it can be expressed by at most 2 distinctive same-orientation supermessages. The same conclusion holds for the message between a receiver proxy and its master. A message between proxies is a non-constrained point-to-point message. Additionally, messages between masters and their proxies on both sender and receiver side each yield at most 1 rerouting (Theorem 2). Finally, an inter-proxy message causes 1 rerouting on the core of a proxy sender and 1 on the core of a proxy receiver. □

By introducing placement and rerouting constraints, supermessages and proxies, we consciously made a decision to potentially "sacrifice" performance (i.e. messages may traverse longer distances and involve reroutings). Nonetheless, this approach causes predictable and deterministic message paths, which allows us to perform a more detailed and less pessimistic analysis (covered later in Section VI-D), and derive tighter worst-case communication delay estimates.

According to Theorems 2-3, all intra- and inter-application traffic of all applications can be expressed with (i) a set of supermessages – $\widehat{\mathcal{M}}$, (ii) a set of proxy-to-proxy messages – $\mathcal{M}_P$, and (iii) a set of reroutings – $\mathcal{R}$, which are all master-independent and known at design time. In order to be able to perform the analysis, the maximum number of occurrences $O(m)$ of each message $m \in (\widehat{\mathcal{M}} \cup \mathcal{M}_P)$ within a minimum inter-arrival period of its application has to be computed. Also, the maximum number of rerouting occurrences $r(d) \in \mathcal{R}$, caused by each dispatcher d on its core within the same interval has to be computed.

2) **Finding the maximum number of occurrences:** As is evident from the previous section, the maximum number of occurrences of both supermessages and reroutings of each application depends on (i) the employed agreement protocol and (ii) the amount of its inter-application traffic. So far, 3 agreement protocols have been proposed (*Master-Slave*, *List* and *Hybrid*) [9]. Since the first one suffers from race conditions, in this paper we focus only on the second and third.

• **List protocol** [9] is an opportunistic agreement protocol; if a master can continue the execution – it does so, without initiating the communication. Otherwise it sends a request to the first neighbouring slave assuming a clockwise orientation. The slave tries to accommodate the execution of the next job instance. If it can, it signals to the old master, receives a context and becomes the new master. Otherwise it passes a request to the next neighbouring slave, also assuming a clockwise orientation. The process repeats until one dispatcher announces the possibility to execute the next job and consequently becomes a new master. The worst-case scenario occurs when the job can be accommodated only by the last traversed dispatcher, and it has to be covered[1]. The downside of this protocol is that a selective scheduling policy is not easily implemented, i.e. the first dispatcher that can execute the job will do so, while maybe better candidates exist within non-traversed dispatchers.

By applying Theorem 2 to every message of the List protocol, we can analytically express the entire protocol as a function of supermessages and reroutings. Subsequently, we can compute the isolation delay $del_I(List)$, the blocking delay $del_B(List)$ and the rerouting delay $del_R(List)$ of one protocol execution. Furthermore, we can obtain the maximum number of occurrences of each supermessage $\widehat{m}$ during one protocol execution $O_P(\widehat{m})$ and during one context transfer $O_C(\widehat{m})$. Finally we can compute the maximum number of reroutings $r_P(d)$ occurring on the core of a dispatcher d, caused by one protocol execution of its application. The analytical steps how to obtain these values are given in Appendix A.

• **Hybrid protocol** [9] is the agreement protocol which consists of two phases. The first phase is initiated when a current master sends a broadcast towards all slave dispatchers. In this way a master requests the information from all slaves about how likely it is that they can accommodate the execution of the next job on their cores. The requests may require the information regarding core utilisation, core temperature, schedulability, etc.[2] Upon receiving the request, each slave queries its kernel for requested information. When the kernel replies, the slave sends the response back to the master. The end of the first phase is reached when the master receives all responses. Then, the master makes an ordered list where all dispatchers are sorted based on the likelihood of accommodating the execution of the next job. If the master is on the top of the list, it continues to be the master. Otherwise, the second phase begins. It is similar to the list protocol, i.e. the master sends a message towards the first slave. However, in this case not to the nearest neighbour in the clockwise orientation, but to the first dispatcher of the generated list. If the slave can accommodate the execution, it requests the context from the master. If not, it passes the request to the next dispatcher from the list. The process repeats until one dispatcher informs the master that it can schedule the job[3]. The worst-case scenario occurs when only the last dispatcher of the ordered list is able to do so. Finally, the master transfers the context to the newly elected slave, which becomes the new master.

Similarly to the List protocol, Theorem 2 can be applied to every message of the Hybrid protocol, and the entire protocol can also be analytically expressed as a function of supermessages and reroutings. By doing so, we can obtain the isolation delay $del_I(Hyb)$, the blocking delay $del_B(Hyb)$ and the rerouting delay $del_R(Hyb)$ of one protocol execution. Additionally, we can compute the maximum number of occurrences of each supermessage $\widehat{m}$ during one protocol execution $O_P(\widehat{m})$ and during one context transfer $O_C(\widehat{m})$. Finally we can compute the maximum number of reroutings $r_P(d)$ occurring on the core of a dispatcher d, caused by one protocol execution of its application. The analytical steps how to obtain these values are given in Appendix B.

• Furthermore, each application may perform **inter-application communication** with other applications. Recall (Theorem 3), that any inter-application message can be expressed as a function of (i) sender's supermessages, (ii) receiver's supermessages, (iii) a proxy-to-proxy message and (iv) reroutings. Thus, by applying Theorem 3 to every inter-application message of one application, we can analytically describe its entire inter-application traffic as a function of the aforementioned constructs. Subsequently, we can obtain the isolation delay $del_I(Send)$, the blocking delay $del_B(Send)$ and the rerouting delay $del_R(Send)$ of the inter-application traffic that an application sends. Similarly, we can obtain the isolation delay $del_I(Rcv)$, the blocking delay $del_B(Rcv)$ and the rerouting delay $del_R(Rcv)$ of the inter-application traffic that an application receives. Additionally, for each supermessage $\widehat{m}$ and inter-proxy message m_{proxy} the maximum number of occurrences as a consequence of the inter-application traffic can be obtained – $O_I(\widehat{m})$ and $O_I(m_{proxy})$, respectively. Finally, the maximum number of reroutings $r_I(d)$ that a dispatcher d induces during the inter-application communication of its application can be computed. The analytical steps how to obtain these values are given in Appendix C.

To summarise, between any two consecutive job releases of an application, its supermessage $\widehat{m}$ can appear $O_P(\widehat{m})$ times during the protocol execution, $O_C(\widehat{m})$ times during the context transfer and $O_I(\widehat{m})$ times for every sent and received inter-application message. Furthermore, each proxy-to-proxy message m_{proxy} may appear $O_I(m_{proxy})$ times due to inter-application traffic. Additionally, the maximum number of reroutings that each dispatcher d induces is equal to the sum of reroutings during a protocol execution and during inter-application communication – $r(d) = r_P(d) + r_I(d)$.

D. Performing the Analysis

Let a be the application under analysis. Its worst-case communication delay (Equation 7), consists of several terms, namely, the isolation delay $del_I(a)$, the blocking delay $del_B(a)$, the rerouting delay $del_R(a)$, the network interference

[1] In this work we assume that, at any time instant, at least one of the dispatchers will be able to accommodate the execution of the next job. Providing such guarantees falls into the domain of LMM schedulability analysis, and it is covered in our work [27].

[2] The policy of the agreement protocol depends on the specific purpose of the system and is immaterial for the discussion in this paper, since we study protocols only as an infrastructure to allow load balancing under LMM.

[3] See Footnote 1

$del_{NI}(a)$ and the rerouting interference $del_{RI}(a)$. Now we describe how to compute the individual terms of Equation 7.

$$del(a) = del_I(a) + del_B(a) + del_R(a) + del_{NI}(a) + del_{RI}(a) \quad (7)$$

• The **isolation delay** (Equation 8). It is equal to the sum of isolation latencies of (i) the protocol execution, (ii) sent inter-application messages and (iii) received inter-application messages. Depending on the agreement protocol of the application under analysis $del_I(Protocol(a)) \in \{del_I(List), del_I(Hyb)\}$.

$$del_I(a) = del_I(Protocol(a)) + del_I(Snd) + del_I(Rcv) \quad (8)$$

• The **blocking delay** (Equation 9) and the **rerouting delay** (Equation 10). They are equal to the sum of blocking and rerouting latencies of (i) the protocol execution and (ii) inter-application traffic. Depending on the selected agreement protocol, $del_B(Protocol(a)) \in \{del_B(List), del_B(Hyb)\}$, and $del_R(Protocol(a)) \in \{del_R(List), del_R(Hyb)\}$.

$$del_B(a) = del_B(Protocol(a)) + del_B(Snd) + del_B(Rcv) \quad (9)$$

$$del_R(a) = del_R(Protocol(a)) + del_R(Snd) + del_R(Rcv) \quad (10)$$

• The **network interference** (Equation 14). First, let us discuss the interference that a higher priority supermessage $\widehat{m}$ of an application $a' \neq a$ can cause to a, between any two consecutive job releases of a' (Equation 11). It is equal to the sum of isolation and blocking latencies (i) while performing the protocol, (ii) while transferring the context and (iii) while performing the inter-application communication.

$$\delta(\widehat{m}) = \overbrace{O_P(\widehat{m}) \times (l_P(\widehat{m}) + b(\widehat{m}))}^{\text{protocol}} + \overbrace{O_C(\widehat{m}) \times (l_C(\widehat{m}) + b(\widehat{m}))}^{\text{context}} + \overbrace{\sum_{\forall m \in \mathcal{M}_S(a') \cup \mathcal{M}_R(a')} O_I(\widehat{m}) \times (l_I(\widehat{m}) + b(\widehat{m}))}^{\text{inter-application traffic}} \quad (11)$$

Notice, that context transfers and inter-application messages may have different sizes than protocol messages, hence their occurrences and traversal latencies have to be computed separately, i.e. $l_P(\widehat{m}) \neq l_C(\widehat{m}) \wedge l_P(\widehat{m}) \neq l_I(\widehat{m})$. Conversely, the blocking delay $b(\widehat{m})$ is not dependent on the message size, but is constant for all occurrences of the supermessage (Equation 2). Thus, $b_P(\widehat{m}) = b_C(\widehat{m}) = b_I(\widehat{m}) = b(\widehat{m})$.

Second, let us discuss the interference that a higher priority inter-proxy message m_{proxy} of an application $a' \neq a$ can cause to a, between any two consecutive job releases of a' (Equation 12). Recall, inter-proxy messages appear only in the inter-application traffic and not in protocols.

$$\delta(m_{proxy}) = O_I(m_{proxy}) \times (l(m_{proxy}) + b(m_{proxy})) \quad (12)$$

Once we have obtained interference delays that any higher priority supermessage and any higher priority inter-proxy message might cause (Equations 11-12), we can compute the total network interference $del_{NI}(a)$ that a suffers during its minimum inter-arrival period. As paths of all messages are deterministic and known at design time, we can take that knowledge into account when performing the analysis. Recall, that in Section V we defined a set $\mathcal{M}_D(m)$ of directly interfering messages of a message m. Those are higher priority messages that share a part of the path with m and hence can preempt it and cause interference. Similarly, here we define a set of directly interfering messages, but of an entire application a, termed $\mathcal{M}_D(a)$. A message belongs to the set $\mathcal{M}_D(a)$ if it is a directly interfering message of any message $m \in \left(\widehat{\mathcal{M}(a)} \cup \mathcal{M}_P(a)\right)$, where $\widehat{\mathcal{M}(a)}$ is the set of supermessages of a and $\mathcal{M}_P(a)$ is the set of inter-proxy messages of a. Formally:

$$\forall m' : \exists m \in \left(\widehat{\mathcal{M}(a)} \cup \mathcal{M}_P(a)\right) \wedge m' \in \mathcal{M}_D(m) \Rightarrow m' \in \mathcal{M}_D(a) \quad (13)$$

Every message $m' \in \mathcal{M}_D(a)$ can cause interference to a. Thus, the maximum network interference that a can suffer, from all its directly interfering messages, is computed by summing up individual interferences that each message $m' \in \mathcal{M}_D(a)$ can cause (Equation 14). Individual terms are obtained by multiplying the maximum interference that m' can induce between any two consecutive job releases of its application a' (computed either by Equation 11 if m' is a supermessage, or by Equation 12 if m' is a proxy message), with the maximum number of inter-arrivals of a'.

$$del_{NI}(a) = \sum_{\forall m' \in \mathcal{M}_D(a)} \left(1 + \left\lceil \frac{T_a - C_{a'})}{T_{a'}} \right\rceil\right) \times \delta(m') \quad (14)$$

• The **rerouting interference** (Equation 16). This is the interference that an application suffers while performing its rerouting operations. Notice that an application can suffer rerouting interference only on cores (tiles) where itself has dispatchers that perform rerouting operations. Let $\mathcal{D}_R(a)$ be a set of dispatchers of a which induce at least one rerouting operation on their respective cores. Formally:

$$\forall d \in \mathcal{D}_a : r_P(d) \neq 0 \vee r_I(d) \neq 0 \Rightarrow d \in \mathcal{D}_R(a) \quad (15)$$

The routing interference of a (Equation 16) is computed by summing up individual interferences that each on-core dispatcher d' might cause to each dispatcher $d \in \mathcal{D}_R(a)$.

$$del_{RI}(a) = \sum_{\forall d \in \mathcal{D}_R(a)} \sum_{\forall d' \in c(d)} \left(1 + \left\lceil \frac{T_a - C_{a'}}{T_{a'}} \right\rceil\right) \times (r_P(d') + r_I(d')) \times l_r \quad (16)$$

$c(d)$ represents the core of a dispatcher d, and l_r denotes the latency of a single rerouting operation. Individual terms are obtained by multiplying the maximum interference that d' can cause between any two consecutive job releases of its application a' with the number of its inter-arrivals.

E. Discussion

The extended version of the previous method (see Section VI-B) does not involve reroutings, and hence does not have rerouting-related components $del_R(a)$ and $del_{RI}(a)$. However, the limitation of this approach is that it treats the path non-determinism in a pessimistic manner and computes the interference on the application-level (see Equation 6). Conversely, the newly proposed analysis employs application placement and rerouting constraints, in order to express the traffic as a function of supermessages, inter-proxy messages and reroutings. This strategy imposes longer message distances and employs a rerouting mechanism, which both additionally contribute to the worst-case communication delay. However, this approach makes message paths deterministic and known at design time. Consequently, the interference component can be computed

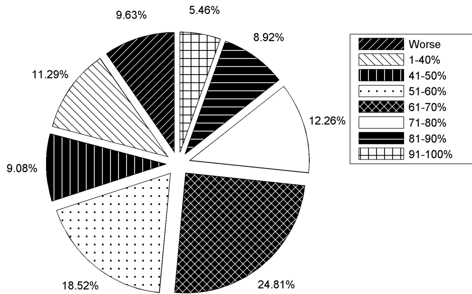

Fig. 5: Overall analysis improvements

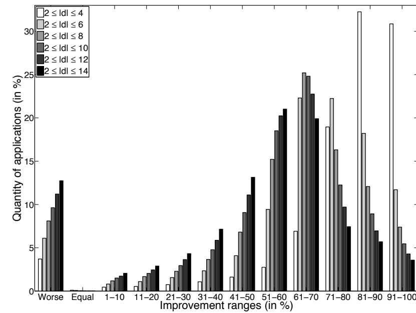

Fig. 6: Improvements wrt dispatchers

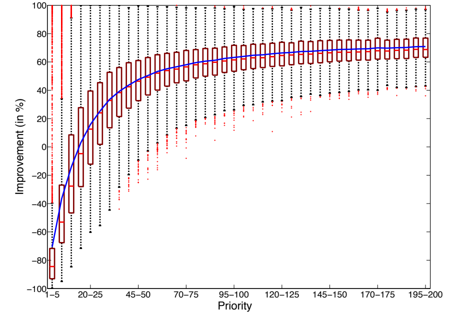

Fig. 7: Improvements wrt priorities

with much less pessimism and hence tighter upper-bounds on the entire worst-case communication delay can be obtained. We back up this claim with the fact that the recent insights into the priority-based wormhole-switched NoCs [30] have proven that the most dominant factor in the worst-case communication delay analysis is the interference component, hence deriving its upper-bound as tight as possible is very important.

VII. Evaluations

We evaluate the proposed approach against the extended version of the existing method [9], presented in Section VI-B, referred to as the *existing method* in the subsequent text. Specifically, we perform the comparison in terms of the analysis and the runtime performance. Since platforms with priority-based wormhole-switched NoCs and flit-level preemptions are still not available, the simulations were performed on the SPARTS [31] simulator. For each experiment we randomly generate application-sets and map them on the platform. Subsequently, for both methods we (i) compute the upper-bound on the worst-case communication delay ($WCCD$) of each application and (ii) perform the simulations to obtain the observed $WCCD$ of each application. Then, we compare the results, and investigate how the trends change with different application parameters, e.g. number of dispatchers, priorities, employed agreement protocols. The analysis and simulation parameters are given in Table I. An asterisk sign denotes a randomly generated value, assuming a uniform distribution.

TABLE I: Analysis parameters

Parameter	Value
Platform size	**10×10**
Application-set size	**200**
Router arbitration + switch latency - l_{sw}	**3 cycles**
Router transfer latency - l_t	**1 cycle**
Rerouting operation latency - l_r	**10.000 cycles**
Mesh width	**16B**
Protocol message size	**1kB**
Context size = Inter-Application message size	**[1-128]* kB**
Minimum inter-arrival period of applications	**[30-1000]* msec**
Probability of inter-application communication	**5 %**
Simulated time	**100 sec**

Experiment 1. *Overall Analysis Improvements*

In this experiment we conducted the overall analytic comparison of the proposed method and the existing one. Specifically, each application had $[2-10]^*$ dispatchers, and was randomly mapped on the grid with arbitrary (rectangular or line-like) shape, assuming dispatcher placement constraints (Constraint 1). Half of the applications executed the List protocol, and the other half Hybrid. Then, the analytic upper-bound on $WCCD$ was obtained for each application of the application-set with both approaches. Finally, we compared the obtained values. The process was repeated for 1000 application-sets.

Figure 5 shows the improvements of the proposed approach over the existing one. In only 9.63% of the cases the proposed approach rendered worse results. This is further investigated in Experiment 3. In the rest of 90.37% scenarios the proposed approach reports improvements. The improvements were expressed in percentages, using the following metric: $imp = \frac{del_{old}(a)-del_{new}(a)}{del_{old}(a)}$, where $del_{old}(a)$ corresponds to the upper-bound on $WCCD$ of an application a obtained by the existing method, and $del_{new}(a)$ with the proposed method. In more than half of the cases the improvements are greater than 50%, which means that a derived upper-bound is at most half the one obtained with the existing approach. Finally, in 5.46% of the scenarios the improvements are above 90%, which corresponds to an estimate which is at most **one-tenth** of the value against which it is compared! That is, $\boldsymbol{del_{new}(a) \leq \frac{1}{10} del_{old}(a)}$.

Experiment 2. *Analysis Improvements wrt to Dispatchers*

In order to test the scalability of the proposed approach we varied the number of dispatchers $|d| = [2-x]^*$ constituting each application. The parameter x was varied in the range $x \in \{4, 6, 8, 10, 12, 14\}$ (see the legend of Figure 6). Assuming a certain range (e.g. $|d| = [2-4]^*$), we generated and randomly mapped all applications. Then, for each application we obtained analytic $WCCD$ upper-bound estimates, computed by (i) the existing method and (ii) the proposed method, compared them with the same metric, and repeated that for 1000 application-sets. The process was performed for every range (Figure 6).

As the number of dispatchers increases, so does the category with worse results and the categories with smaller improvements (until 60%), while other categories with more significant improvements (above 60%) decrease. The explanation is twofold. First, assuming the proposed approach, more dispatchers cause more reroutings. This in turn causes more significant rerouting interferences, which have an impact on the derived worst-case delays. Conversely, in the existing approach reroutings do not occur. Additionally, more dispatchers cause more messages, leading to more significant and complex message interfering scenarios. In such cases, making an assumption that every higher priority message existing within the network will indeed cause interference might not be too pessimistic. Thus, as the number of dispatchers increases, the existing method

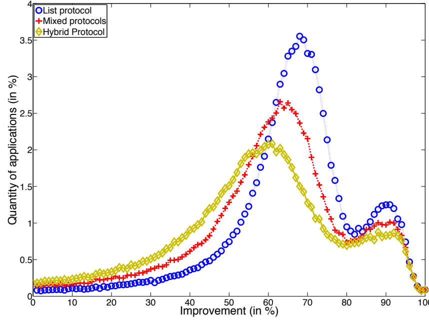

Fig. 8: Improvements wrt protocols

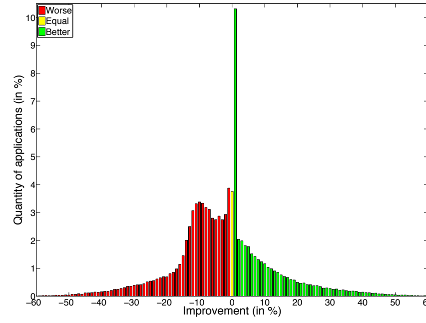

Fig. 9: Overall performance comparison

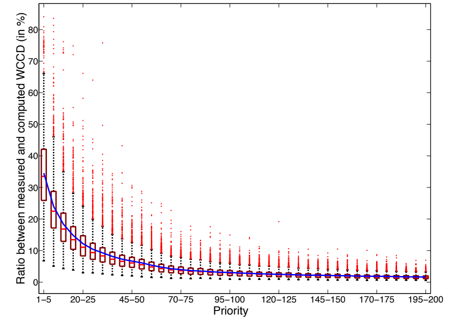

Fig. 10: Analysis tightness

becomes less pessimistic and hence improvements caused by the proposed method over it are slowly decreasing.

Experiment 3. *Analysis Improvements wrt to Priorities*

In this experiment we investigated how the improvements of the proposed method over the existing one change with application priorities. This experiment also helped us to recognise and investigate the cases where the proposed method underperformed. The values obtained in Experiment 1 were used, but the comparison between the approaches was additionally performed per-priority and depicted in Figure 7.

Improvements were measured with the same metric, while the cases where the proposed approach underperformed were presented with the transposed metric: $imp = \frac{del_{old}(a) - del_{new}(a)}{del_{new}(a)}$. It is evident that the proposed approach performs worse for applications with higher priorities (smaller numbers on the x-axis of Figure 7). As these applications do not suffer significant interference, the additional delay in the proposed approach is caused by reroutings and longer message distances (i.e. traversal constraints expressed by Constraint 2). As priorities decrease, the interference becomes a more dominant term in the derived $WCCD$ upper-bounds, hence the penalty of the proposed approach slowly decays. At the priority level 15, both the approaches provide similar results. Any additional decrease in the priority favours more the proposed method, where improvements report logarithmic growth. On the far right end of the domain the average improvements of the proposed approach asymptotically converge towards 70%.

Experiment 4. *Analyis Improvements wrt to Protocols*

In this experiment we compared the approaches with an emphasis on the employed agreement protocols. The comparison was performed for 3 different scenarios: (i) all applications are utilising the List protocol, (ii) all applications are utilising the Hybrid protocol, (iii) half of the applications are utilising each protocol. For each application the $WCCD$ upper-bound estimate were obtained with both approaches. The process was repeated for 1000 application-sets. The results are plotted in Figure 8, where the improvements of the proposed approach over the existing one were expressed by the same metric used in the previous experiments.

The conclusions are similar to that of Experiment 2. The Hybrid protocol involves more messages, which in the proposed approach induce more reroutings. Hence, the biggest improvements are reported for the List protocol, then for mixed protocols, while the least improvements occur with the Hybrid protocol. Additionally, as the improvement metric is based on the ratio, similarly to the logarithmic scale, each additional improvement percentage covers larger part of the domain; e.g. for improvements of $49 - 50\%$ the following holds: $del_{old}(a) \in \{1.96 \times del_{new}(a), 2 \times del_{new}(a)\}$, while for improvements of $89 - 90\%$ the following holds: $del_{old}(a) \in \{9.09 \times del_{new}(a), 10 \times del_{new}(a)\}$. Due to that fact, improvement ranges around 90% cover large parts of the domain, and cause small local maximums, visible in Figure 8.

Experiment 5. *Overall Performance Comparison*

The purpose of this experiment is to compare the runtime performance of the proposed and the existing approach. In order to do that, we have used application-sets generated for Experiment 1 and performed the simulations. The execution of each application-set was simulated for 100 seconds with both approaches. Within each approach $WCCD$ was measured for each application. Consequently, the obtained values were compared and the results are presented in Figure 9.

Since the proposed approach induces longer message distances and employs the rerouting mechanism, we were intuitively expecting that it will systematically suffer a significant runtime performance penalty, when compared with the existing method. However, we reached the surprising conclusion: **not only was the penalty negligible in almost all cases, but also for almost 40% of the scenarios the proposed approach outperformed the existing one!** We interpret these unexpected findings in the following way. Unpredictable message paths may lead to corner cases where a traffic becomes heavily concentrated in certain links of the grid, resulting in significant contentions and (almost) unbounded interference delays. This coincides with the conclusions from our previous work [30], that network interference indeed is the most dominant component in $WCCD$, and also reflects the importance of having predictable message-paths, which additionally validates the contributions of this work. This further implies that the overall efficiency of the system is heavily dependent on the application-mapping process, and we see this area as a potential topic for the future work.

Experiment 6. *Analysis Tightness*

Finally, in this experiment we tested the tightness of the proposed approach. Specifically, for each application of the application-set we compared the analytically computed upper-

bound estimate on its $WCCD$ against the $WCCD$ value obtained via simulations. The input parameters are identical to that of Experiment 1 and Experiment 5. In order to get a better insight into the ratios between obtained values, we have grouped applications into categories, according to their priorities. Figure 10 illustrates the findings.

The highest observed ratio between the measured $WCCD$ and the analytically computed upper-bound estimate was around 85%, which suggests that the analysis is correct and renders tight estimates. Furthermore, the results demonstrate that the average ratio is almost 35% for applications with the highest priority (lower numbers). As the priorities keep decreasing, so does the ratio, which asymptotically converges towards 2% for applications with the lowest priority. The explanation is as follows. As the priority decreases, the analysis considers more and more complex interference (worst-case) scenarios, which are less and less likely to occur during simulations. Given our findings, that the chance of suffering the worst-case scenario by lower-priority applications is small, and that in many practical scenarios their occasional missed deadlines are tolerable, several questions can be raised. For instance, **is the worst-case analysis the right approach to treat the lower-priority applications?** Or should some other (e.g. probabilistic) techniques be applied? We see these questions as starting points for our future work.

VIII. Conclusions and Future Work

The Limited Migrative Model (LMM) is a promising step towards scalable and predictable many-cores, which are essential prerequisites for their integration in the real-time embedded domain. Assuming LMM, we analysed the worst-case communication delays of individual applications. First, we extended LMM to accommodate the inter-application traffic. Then, we enhanced the existing analysis [9], so as to make it applicable to the extended model. Finally, we proposed a novel approach which potentially "sacrifices" the performance in order to gain in predictability (i.e. determinism in message paths). We compared the methods both in terms of the analysis and the performance. The experiments show that the proposed approach not only renders tighter upper-bound estimates in more than 90% of the cases, but also demonstrates a comparable runtime performance, which reflects the importance of having deterministic traffic routes.

The future work has already been discussed throughout the paper; we plan to develop an application mapping approach which will use the presented analysis, together with an LMM schedulability analysis (e.g. [27]), as feasibility tests. Also, providing an alternative (probabilistic?) method to treat lower-priority workload is a challenge with an increasing importance.

References

[1] M. Lundstrom, "Moore's law forever?" *Science*, 2003.

[2] Intel, *The Single-chip Cloud Computer*, www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html.

[3] Tilera, *TILE64™ Processor*, www.tilera.com/products/processors/TILE64.

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

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

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

[7] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: A new os architecture for scalable multicore systems," in *SOSP*, 2009.

[8] R. Kumar, T. G. Mattson, G. Pokam, and R. van der Wijngaart, "The case for message passing on many-core chips," in *Multiprocessor System-on-Chip*. Springer, 2011, pp. 115–123.

[9] B. Nikolić and S. M. Petters, "Towards network-on-chip agreement protocols," in *12th EMSOFT*, 2012.

[10] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): the case for a scalable operating system for multicores," *SIGOPS Oper. Syst. Rev.*, 2009.

[11] M. D. Y. Li and R. West, "Quest-v: A virtualized multikernel for high-confidence systems," Tech. Rep., http://www.cs.bu.edu/~richwest/quest.html.

[12] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, 1973.

[13] K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," in *30th RTSS*, 2009.

[14] S. Kato, N. Yamasaki, and Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors," in *21st ECRTS*, 2009.

[15] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," in *25th annual ACM symposium Theory computing*, New York, NY, USA, 1993.

[16] P. L. Holman, "On the implementation of pfair-scheduled multiprocessor systems," Ph.D. dissertation, The University of North Carolina at Chapel Hill, North Carolina, United States, 2004.

[17] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *47th DAC*, 2010.

[18] B. Kim, J. Kim, S. Hong, and S. Lee, "A real-time communication method for wormhole switching networks," in *1998 Int. Conf. Parall. Processing*, Aug 1998.

[19] D. Dasari, B. Nikolić, V. Nelis, and S. M. Petters, "Noc contention analysis using a branch and prune algorithm," *Trans. Emb. Comput. Syst.*, 2013.

[20] T. Ferrandiz, F. Frances, and C. Fraboul, "A method of computation for worst-case delay analysis on spacewire networks," in *IEEE Int. Symp. Industrial Emb. Syst.*, 2009.

[21] ——, "Using network calculus to compute end-to-end delays in spacewire networks," *SIGBED Rev.*, 2011.

[22] Y. Qian, Z. Lu, and W. Dou, "Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip," in *Int. Symp. Netw.-on-Chip*, 2009.

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

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

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

[26] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Int. Symp. Netw.-on-Chip*, 2008.

[27] B. Nikolić, K. Bletsas, and S. M. Petters, "Priority assignment and application mapping for many-cores using a limited migrative model," Tech. Rep., available at: http://www.cister.isep.ipp.pt/people/Borislav+Nikolic/publications/.

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

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

[30] B. Nikolić, H. I. Ali, S. M. Petters, and L. M. Pinho, "Are virtual channels the bottleneck of priority-aware wormhole-switched noc-based many-cores?" in *21th RTNS*, 2013.

[31] B. Nikolić, M. A. Awan, and S. M. Petters, "SPARTS: Simulator for power aware and real-time systems," in *8th IEEE ICESS*, 2011.

Appendix A
List Protocol

Before going into details on how to analytically express the List protocol as a function of supermessages and reroutings,

we introduce two constructs which will aid us in that cause.

Definition 3 (Simple/Complex message). *An intra-appli-cation message is called* simple message - m^s *if it can be expressed with a single supermessage. Otherwise, it is called* complex message - m^c.

An illustrative example of Defnition 3 is given in the bottom right part of Figure 3.

Theorem 4. *The List protocol (including a context transfer) of an application with $|d|$ dispatchers can be expressed as a linear combination of supermessages, where clockwise supermessages can exist at most $|d|+1$ times, and counter-clockwise can exist at most twice.*

Proof. Proven directly. As neighbouring dispatchers always share the same edge, all messages exchanged by neighbouring dispatchers are *simple messages*. Thus, they can be expressed by only one clockwise supermessage and do not involve reroutings. Unless the exact position of neighbouring dispatchers on the application shape is known, it is not possible to deduce which one of clockwise supermessages will be used. Therefore, each of clockwise supermessages might appear in all $|d|-1$ messages, starting from the master until reaching the last slave. The message from the last slave to the master, and the following context transfer are not necessarily exchanged by neighbouring dispatchers, thus for the worst-case have to be considered as *complex messages*. Moreover, these two messages may be of an arbitrary orientation, thus any supermessage may appear once in each of them (Theorem 2). □

Equations 17-19 express the maximum number of occurrences of each supermessage during one protocol execution. As explained in Section VI-D, contexts and protocol messages may have different sizes – $l_P(\widehat{m}) \neq l_C(\widehat{m})$, thus respective supermessage occurrences have to be counted separately.

$$O_P(\widehat{m_{cw1}}) = O_P(\widehat{m_{cw2}}) = |d| \quad (17) \qquad O_P(\widehat{m_{cc1}}) = O_P(\widehat{m_{cc2}}) = 1 \quad (18)$$

$$O_C(\widehat{m_{cw1}}) = O_C(\widehat{m_{cw2}}) = O_C(\widehat{m_{cc1}}) = O_C(\widehat{m_{cc2}}) = 1 \quad (19)$$

Now we obtain the maximum number of reroutings as a consequence of one protocol execution (Theorem 5).

Theorem 5. *The List protocol of a $|d|$-dispatcher application can involve at most 2 reroutings.*

Proof. Proven directly. Until reaching the last slave all messages are simple messages, can be expressed with one supermessage and hence do not involve reroutings (Theorem 2). A message from the last slave to the master and a subsequent context transfer can be complex messages and, each can contribute with one rerouting (Theorem 2). Thus, the maximum number of reroutings – $R(List)$ is 2 (Equation 20). □

$$R(List) = 2 \quad (20) \qquad r_P(d_C) = 2 \quad (21)$$

Note, that only a dispatcher d_C positioned in the corner of an application shape may accommodate reroutings. The maximum rerouting overhead that d_C may induce, noted down as $r_P(d_C)$, occurs when both reroutings are performed by it (Equation 21). Also note, a 4-dispatcher application does not involve reroutings, as all corner dispatchers are mutually reachable via supermessages. In such cases, $R(List) = 0$.

Now we can derive the worst-case isolation delay of the List protocol – $del_I(List)$ (Equation 22). Until reaching the last slave, all requests are simple messages, thus are expressed by only one clockwise supermessage (the first term in Equation 22). The response from the last slave and the context transfer are complex messages of arbitrary orientation, and by Theorem 2 are expressed as two same-orientation supermessages. Notice, that each supermessage engaged in protocols has the same size and traverses the path of the same length, thus the isolation latency of each protocol-related supermessage is the same, i.e. $l_P(\widehat{m_{cw1}}) = l_P(\widehat{m_{cw2}}) = l_P(\widehat{m_{cc1}}) = l_P(\widehat{m_{cc2}}) = l_P(\widehat{m})$. The same holds for context transfers, i.e. $l_C(\widehat{m_{cw1}}) = l_C(\widehat{m_{cw2}}) = l_C(\widehat{m_{cc1}}) = l_C(\widehat{m_{cc2}}) = l_C(\widehat{m})$, however, recall that $l_P(\widehat{m}) \neq l_C(\widehat{m})$.

$$del_I(List) = \overbrace{(|d|-1) \times l_P(\widehat{m})}^{\text{until the last slave}} + \overbrace{2 \times l_P(\widehat{m})}^{\text{last slave to master}} + \overbrace{2 \times l_C(\widehat{m})}^{\text{context transfer}} \quad (22)$$

By following the aforementioned logic, we compute the blocking and the rerouting delay of the List protocol – $del_B(List)$ and $del_R(List)$, respectively (Equations 23-24). The blocking delay of a message – $b(\widehat{m})$ is not dependent on the message size (see Equation 2), hence is the same for protocol and context transfers, i.e. $b_P(\widehat{m}) = b_C(\widehat{m}) = b(\widehat{m})$.

$$del_B(List) = ((|d|-1) + 2 + 2) \times b(\widehat{m}) \quad (23) \qquad del_R(List) = R(List) \times l_r \quad (24)$$

Appendix B
Hybrid Protocol

Theorem 6. *The Hybrid protocol (including a context transfer) of a $|d|$-dispatcher application can be expressed as a linear combination of supermessages, where each supermessage can exist at most $3 \times |d| - 1$ times.*

Proof. In the Hybrid protocol messages are not necessarily exchanged by neighbouring dispatchers. Therefore, in order to analyse the worst-case, all messages have to be considered as complex messages of arbitrary orientation. By Theorem 2, each message can involve 2 supermessages of the same orientation. Thus, every supermessage can exist once within each protocol message. The first phase involves requests sent to all $|d|-1$ slaves and their $|d|-1$ replies, rendering at most $2 \times (|d|-1)$ appearances of each supermessage. During the second phase $|d|$ messages are exchanged until reaching the master, resulting in additional $|d|$ appearances of each supermessage. Finally, the context transfer yields one additional occurrence of each supermessage. □

The maximum number of occurrences of supermessages during one protocol execution and during one context transfer are given in Equations 25-26.

$$O_P(\widehat{m_{cw1}}) = O_P(\widehat{m_{cw2}}) = O_P(\widehat{m_{cc1}}) = O_P(\widehat{m_{cc2}}) = \overbrace{2 \times (|d|-1)}^{\text{the first phase}} + \overbrace{|d|}^{\text{the second phase}} \quad (25)$$

$$O_C(\widehat{m_{cw1}}) = O_C(\widehat{m_{cw2}}) = O_C(\widehat{m_{cc1}}) = O_C(\widehat{m_{cc2}}) = 1 \quad (26)$$

Let us now compute the maximum number of reroutings that happen within one protocol execution (Theorem 7).

Theorem 7. *The Hybrid protocol of a $|d|$-dispatcher application can involve at most $3 \times |d| - 1$ reroutings.*

Proof. Proven directly. Each message is treated as a complex message and, by Theorem 2, can cause at most one rerouting. Thus, the maximum number of reroutings – $R(Hyb)$ is equal to the maximum number of messages, hence in total $3 \times |d| - 1$ reroutings might occur (Equation 27). □

$$R(Hyb) = 3 \times |d| - 1 \quad (27) \qquad r_P(d_C) = 3 \times |d| - 1 \quad (28)$$

A rerouting can occur only on a corner dispatcher d_C. The worst-case rerouting overhead that d_C can induce, noted down as $r_P(d_C)$, occurs when all reroutings are performed on its core (Equation 28). Like in the List protocol, a 4-dispatcher application involves no reroutings: $R(Hyb) = 0$.

Let us now derive the worst-case isolation delay of the Hybrid protocol – $del_I(Hyb)$ (Equation 29). Every message has to be treated as a complex message, thus is represented as the sum of two same-orientation supermessages. Similarly, protocol messages and context transfers are treated separately, because $l_P(\widehat{m}) \neq l_C(\widehat{m})$.

$$del_I(Hyb) = \overbrace{(3|d| - 2) \times 2 \times l_P(\widehat{m})}^{\text{first and second phase}} + \overbrace{2 \times l_C(\widehat{m})}^{\text{context transfer}} \quad (29)$$

By following the same logic, the blocking delay $del_B(Hyb)$ and the rerouting delay $del_R(Hyb)$ of the Hybrid protocol are given in Equations 30-31, respectively.

$$del_B(Hyb) = (3|d|-1) \times 2 \times b(\widehat{m}) \quad (30) \qquad del_R(Hyb) = R(Hyb) \times l_r \quad (31)$$

Appendix C
Inter-application Traffic

Theorem 8. *Any inter-application message, which is, by applying Theorem 3, expressed as a function of supermessages and the inter-proxy message, can yield at most one occurrence of (i) each supermessage and (ii) the inter-proxy message.*

Proof. Follows directly from Theorems 2-3. □

The implications of Theorem 8 are that each sent and received inter-application message triggers one occurrence of the inter-proxy message and each of the supermessages (Equation 32). As different inter-application messages can have different sizes, we do not sum up their occurrences, but instead treat them independently.

$$O_I(\widehat{m_{cw1}}) = O_I(\widehat{m_{cw2}}) = O_I(\widehat{m_{cc1}}) = O_I(\widehat{m_{cc2}}) = O_I(m_{proxy}) = 1 \quad (32)$$

Let us now obtain the maximum number of reroutings induced by one inter-application message.

Theorem 9. *Any inter-application message can involve at most 2 reroutings on the sender's side, of which at most one can appear on any core.*

Proof. Proven directly. Consider an inter-application message from a master sender to its proxy dispatcher d_{PS}. By Theorem 2 it involves one rerouting at the intermediate corner dispatcher d_C which is not d_{PS} since it is the destination of that message. When d_{PS} is reached, one additional rerouting occurs before the inter-proxy message is forwarded to the receiver proxy d_{PR}. Perceived from a sender's perspective, at most 2 reroutings may occur, one performed on d_{PS}, and the other on any other corner-placed dispatcher d_C. □

The proof for the receiver's side is very similar and is therefore omitted. Now, the maximum number of reroutings induced by all sent inter-application traffic of an application a is given in Equation 33, while Equation 34 holds for all received traffic. Recall, $\mathcal{M}_S(a)$ and $\mathcal{M}_R(a)$ have been introduced in Section IV-B and denote sets of sent and received inter-application messages of the application a, respectively. Equation 35 presents the maximum number of reroutings occurring at the same dispatcher.

$$R(Snd) = 2 \times |\mathcal{M}_S(a)| \quad (33) \qquad R(Rcv) = 2 \times |\mathcal{M}_R(a)| \quad (34)$$

$$r_I(d_C) = r_I(d_{PS}) = r_I(d_{PR}) = |\mathcal{M}_S(a)| + |\mathcal{M}_R(a)| \quad (35)$$

Let us now compute the worst-case isolation delay of inter-application traffic. During the analysis, we decompose an inter-application message from an application a to an application a' into 2 disjoint segments: the responsibility of a is to deliver the message to the proxy of a', while delivering it to its master is the responsibility of a'. The isolation delay of all sent messages from the master to proxies of all receivers is described with Equation 36. According to Theorem 2, each message between the master sender and its proxy may involve two supermessages. As already stated (Section VI-D), the traversal latencies of supermessages related to inter-application messages have to be computed separately, as they may have sizes different than protocol messages, thus i.e. $l_P(\widehat{m}) \neq l_I(\widehat{m})$.

$$del_I(Snd) = \sum_{\forall m \in \mathcal{M}_S(a)} (\overbrace{2 \times l_I(\widehat{m})}^{\text{master to proxy}} + \overbrace{l(m_{proxy})}^{\text{inter proxy}}) \quad (36)$$

Similarly, we compute blocking delay and the rerouting delay of all messages from a master sender to proxies of all receivers (Equations 37-38).

$$del_B(Snd) = \sum_{\forall m \in \mathcal{M}_S(a)} (2 \times b(\widehat{m}) + b(m_{proxy})) \quad (37)$$

$$del_R(Snd) = R(Snd) \times l_r \quad (38)$$

Equations 39 - 41 describe the communication perceived from a receiver's perspective.

$$del_I(Rcv) = \sum_{\forall m \in \mathcal{M}_R(a)} \overbrace{2 \times l_I(\widehat{m})}^{\text{proxy to master}} \quad (39)$$

$$del_B(Rcv) = \sum_{\forall m \in \mathcal{M}_R(a)} 2 \times b(\widehat{m}) \quad (40)$$

$$del_R(Rcv) = R(Rcv) \times l_r \quad (41)$$