Modeling High-Performance Wormhole NoCs for Critical Real-Time Embeddedd Systems

Miloš Panić*,†, Carles Hernandez†, Eduardo Quiñones†, Jaume Abella†, Francisco J. Cazorla†,‡
*Universitat Politècnica de Catalunya (Spain) †Barcelona Supercomputing Center (Spain)
‡Spanish National Research Council (IIIA-CSIC) (Spain)

***Abstract*—Manycore chips are a promising computing platform to cope with the increasing performance needs of critical real-time embedded systems (CRTES). However, manycores adoption by CRTES industry requires understanding task's timing behavior when their requests use manycore's network-on-chip (NoC) to access hardware shared resources. This paper analyzes the contention in wormhole-based NoC (wNoC) designs – widely implemented in the high-performance domain – for which we introduce a new metric: *worst-contention delay* (WCD) that captures wNoC impact on *worst-case execution time* (WCET) in a tighter manner than the existing metric, *worst-case traversal time* (WCTT). Moreover, we provide an analytical model of the WCD that requests can suffer in a wNoC and we validate it against wNoC designs resembling those in the Tilera-Gx36 and the Intel-SCC 48-core processors. Building on top of our WCD analytical model, we analyze the impact on WCD that different design parameters such as the number of virtual channels, and we make a set of recommendations on what wNoC setups to use in the context of CRTES.**

I. Introduction

Manycore chips can accommodate high task counts in a single hardware device which helps reducing size, weight and power costs in Critical Real-Time Embedded Systems (CRTES). The deployment of manycores as baseline computing platform in CRTES requires a means for the safe consolidation of multiple CRTES applications on the same chip. In that respect, one of the stumbling blocks in the manycore adoption in CRTES is understanding how manycore internal complexity affects tasks' timing behavior. The interconnection network is, arguably, one of the manycore shared resources with highest impact on timing. Unlike multicores, which use a centralized interconnect (e.g. a bus) to access hardware shared resources, manycores implement networks-on-chip (NoC). In NoCs, requests are arbitrated in a distributed manner at various routers severely complicating timing analysis.

In the high-performance domain, wormhole NoCs (wNoCs) are well understood [9] [12] and used in several Commercial-Off-The-Shelf (COTS) products [30] [36]. The high-throughput and low-hardware cost features of wNoCs make them attractive for CRTES as an alternative to real-time customized networks whose adoption in commercial products is harder to achieve. In this respect, this paper addresses the problem of whether high-performance wNoC designs can be used to consolidate, in a trustworthy manner, multiple CRTES applications into a single manycore. This requires providing high-performance and isolation among tasks so that time composable WCET estimates [27] (independent of the load that co-running tasks put on the NoC) can be derived.

We take time composability as a premise in CRTES design since it enables two fundamental properties to reduce system development and deployment costs: incremental development and incremental verification of integrated systems (e.g. IMA [4], [37], AUTOSAR [5]). During system development, the ability to incrementally integrate applications without the need of regression tests to validate the timing properties of already-integrated applications heavily reduces integration costs. At system deployment, the ability to update functions and their associated software, without the need for re-analyzing and re-certifying the system, is vital in domains like space where systems operate during dozens of years and whose functionality is usually updated once deployed.

We propose an analytical model that captures the impact of the different wNoC design choices and parameters on WCET estimates. Our goal is to adhere to existing COTS wNoC designs without the need of adding extra hardware support. In particular, we make the following contributions:
(1) We introduce worst-contention delay (WCD) as a new metric to accurately capture the impact of wNoC inter-task interferences on WCET estimates (Section III). WCD takes into account the pipelined behavior of wNoCs, leading to tighter WCET estimates compared to the ones obtained with the Worst-Case Traversal Time (WCTT) [29] [19] [28] [8].
(2) We provide a taxonomy of wNoC design parameters (Section IV), identifying those that have to be fixed in order to provide trustworthy and composable WCD bounds; and those where some flexibility is allowed. We show that the default values for some of the latter set of parameters are configured to improve average performance, increasing WCD bounds.
(3) We derive an analytical model for time-composable WCD bounds in a wNoC (Section V), covering a vast set of parameters including flits-per-packet, number of virtual channels and queue size in the router[1]. Our model achieves high coverage of existing COTS high-performance wNoC designs. We discuss static virtual channel allocation and show that it has to be applied smartly to reduce WCD bounds. Otherwise, it can result in an increase in WCD bounds, e.g, using virtual

[1]We consider arbitration, routing and virtual-channel allocation policies, to be configurable from software similarly to the way cache replacement is currently adjustable in high-performance architectures. This is in contrast to hardware proposals that require global changes like, new signals among routers and nodes, different flow-control, global clocks or the like.

channels to separate the traffic generated by applications under different criticality levels increases WCD bounds and hence WCET estimates. Further, in Section VI, we discuss the impact of various wNoC parameters on system design.

(4) We assess the accuracy of our analytical model on two wNoC setups resembling the ones deployed in real processors: the Tilera-Gx36 [36] and the 48-core Intel SCC [30] (Section VII). In all cases, our WCD estimates tightly upperbound the measured contention delay values with up to 5% overestimation on average. Further, we show that on average, WCD bounds are 2.7x and 2.94x lower than WCTT bounds for the Tilera-Gx36 and the Intel SCC setup respectively.

Overall, our analysis shows that simple but effective design and configuration choices make efficient use of wNoCs in CRTES possible.

II. Background

In CRTES, there are two main ways to handle contention among accesses to shared hardware resources, including NoCs. First, the NoC contention is accounted as part of the WCET estimation process by deriving a time composable bound of the *worst-case traversal time* (WCTT). WCTT defines the longest time a request could take since the moment it is injected in the NoC by a source node until it is delivered to the destination node. Alternatively, NoC contention delay that a task suffers can be handled as part of the worst-case response time analysis by adding to the task's execution time in isolation the contention generated by the flows of its co-running tasks – which are assumed known at this stage.

Each approach has its own pros and cons: while the latter enables deriving tighter estimates, since it builds upon the knowledge of interference generated by the tasks in the observed task set, it violates time composability. The former, which is the focus of this paper, maintains time composability at the expense of higher WCET estimates.

In the next section we propose the use of Worst-Contention Delay (WCD) instead of WCTT as a way to provide tighter WCET estimates for the tasks running in the wNoC based manycore processor.

III. Contention Delay: A New Metric to account for the impact of NoC on WCET

Given a task under analysis, we call *contention delay* (CD) the delay caused by the *other* co-running tasks in the access to the shared NoC. As an alternative to WCTT we introduce a new metric, called Worst-Contention Delay (WCD), that captures in a tight manner the impact that accesses to the NoC have on programs' execution time and WCET. WCD stands for the highest impact that a request may have on WCET due to contention in the NoC. It stems from the appreciation that requests can suffer two types of delays: intra-task delay (*atd*) that is caused among requests coming from the same core; and inter-task delay (*etd*) that covers the delay that one request from a core causes on the request of a different core.

We illustrate the difference between WCD and WCTT with the example in Figure 1. Figure 1(a) shows a simple NoC connecting three cores, out of which one is the destination core (c_2). Our focus is determining the delays suffered by the requests from core c_0 when accessing the NoC. An arbiter, which implements round-robin policy, handles requests coming from c_0 and c_1, with separate buffers to handle the requests of c_0 and c_1.

The time diagram in Figure 1(b) shows the actual traversal time and the actual contention delay suffered by requests r_0^0 - r_0^4 sent from c_0 (upper time diagram) and r_1^0 - r_1^4 sent from c_1 (lower time diagram with grey background) when cores inject packets at the maximum rate. In absence of interference, we assume that a request takes 1 cycle to traverse the router, and that buffers can store up to 2 requests. In the time diagram, I stands for the cycle when the request is transferred from c_0 to the buffer and C_X the cycle when the request is sent from the router (eXits) to the target node c_2. Ba corresponds to cycles when the request is in the buffer but not at the top, hence suffering *atd*. Likewise, Be corresponds to cycles when the request is at the top of the buffer and hence suffering *etd* since the arbiter grants access to c_1.

We assume that first request of c_0 lost the arbitration in Cycle 1 so that it has to wait for a request of c_1 to traverse the router. We observe that request r_0^0 only suffers *etd*. r_0^1 enters in the second entry of the buffer in cycle 1 (cyc_1), in cyc_2 it suffers *atd* and reaches the top of the buffer in cyc_3 where it suffers a cycle of *etd*. We observe a similar behavior for other requests, with the difference that when there are two requests in the buffer, c_0 has to wait until one is released before sending another request.

The two columns on the right of the time diagram show the interference cycles for traversal time (cTT) and for contention delay (cCD) metrics. For the traversal time the interference cycles suffered by r_0^0 - r_0^4 are $2, 3, 4, 4, 4$ respectively. Meanwhile for the contention delay they are $1, 1, 1, 1, 1$. The key appreciation is that with traversal time the interference accounted for each request covers both *atd* (Ba) and *etd* (Be). *However, the impact of Ba for one request is already accounted as part of the Be of another request*. For instance, the *atd* suffered by r_0^2 in cyc_3 is the *etd* suffered by r_0^1. Overall, the main problem with traversal time is that it doesn't capture well the pipelined behavior of the NoC and that the same cycle can be accounted several times as contention, either intra- or inter-task, in different requests. Contention delay instead, only focuses on inter-task contention and does not over-account *atd* and *etd* cycles. Therefore, considering $WCTT$ as the extra delay that each request suffers due to contention in the NoC can be (very) pessimistic. Instead, WCD provides tighter estimates since it prevents requests to account for multiple *atd* interference delays as shown in Figure 1. Our results in Section VII show that for a wide range of NoCs *WCD* tightly and trustfully upperbounds the impact of inter-task interferences in wNoCs.

Properties. WCD shows a few interesting properties:

i) Larger buffers increase the *atd* since a request has to potentially wait for a higher number of requests coming from the same core. This translates on the fact that larger buffers

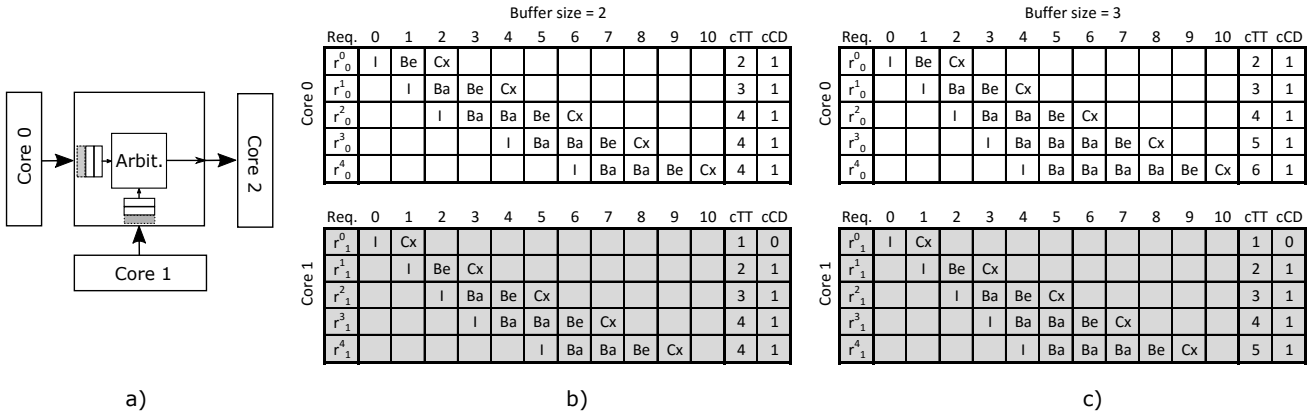


Fig. 1. Simple router and the impact of *atd* and *etd* on traversal time and contention delay.

lead to higher WCTT. This is counterintuitive since the more the resources of the wNoC, in the form of larger buffers, the worse the WCTT is. WCD is immune to this as it is not affected by the buffer size, which enables use of wNoCs with larger buffers in real-time domain with benefits for average performance of the wNoC.

Time diagrams in Figure 1(c) show the behavior or previously discussed sequences of requests from cores c_0 and c_1, if we increase the sizes of input buffers in the router to 3. This change doesn't affect execution time of the sequence nor contention delay. However, it shows an increase in traversal time for some requests, e.g. for r_0^4 it grows from 4 to 6.

ii) Solutions based on limiting the injection rate at hardware [19] or software-level [20] effectively reduce *atd* since requests are injected into the wNoC at a lower rate and so fewer conflicts occurs. In the extreme case, these technique can prevent flows from having more than one packet in any single router, completely removing the *atd* but jeopardizing the wNoC utilization. While this reduces the problem of *atd* accounting for WCTT, WCD completely removes *atd* providing tighter bounds, without any impact on average performance.

iii) WCD leads to tighter WCET estimates than WCTT since the *atd* a request suffers occurs in parallel to the *etd* for other requests, which is already captured by WCD.

Assumptions. *WCD* applies to processors free of timing anomalies such that increasing the local delay suffered by any request leads to an increase in execution time by at most the magnitude of the local delay. In particular, by increasing contention delay by d cycles, execution time grows by up to d cycles. Further, WCD applies to network designs implementing back pressure flow-control policies i.e. NoC designs with no packet loss such as wormhole and virtual cut-through [9]. WCD works for work-conserving policies such as round-robin so that links are never left idle if there are pending requests.

IV. NoC Parameters Taxonomy

This section presents a taxonomy of wNoC parameters. We consider a mesh network topology as it is the most common topology used in wNoCs, though the analytical model presented in this paper also applies to other network topologies (e.g. torus) by simply varying some parameters such as the number of ports per node. Table I lists the symbols and the corresponding description used in the rest of the paper. In this paper we distinguish between the WCD, which corresponds to the actual worst contention delay a NoC request may suffer due to interferences with other requests, and the $\overline{WCD}$, which corresponds to an upper-bound of the actual WCD derived by our analytical model.

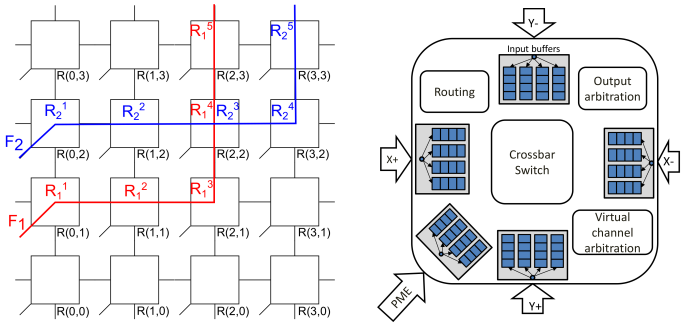
TABLE I
SUMMARY OF MAIN SYMBOLS USED

Symbol	Description
VC	Virtual Channel
$\overline{WCD}$	Time-composable upper bound to contention delay
F_i	Packet stream traversing the same source-destination route and requiring the same grade of service along the path.
H_i	Number of hops in a flow F_i
R_i^j	Router (hop) j in a flow F_i (see Figure 2(a))
r_i^k	Packet (request) k in a flow F_i
L^{flits}	Number of flits of a packet
P_i^j	Number of ports in router R_i^j
NR_i^j	Number of queues that can potentially contend for an output port that F_i is targeting at R_i^j
$\omega(i,j)$	Function that returns the index x of the worst possible destination flow F_x that starts at the hop R_i^{j+1} and reaches the worst possible destination in terms of indirect blocking of packets of flow F_i

A. Wormhole mesh NoC fundamentals

In our reference *NxM* mesh wNoC configuration, depicted in Figure 2(a), each node comprises a *PME* (Processor/Memory element) and a router that communicates with the rest of nodes. The PME can be either a processor core, a cache memory, main memory, I/O, etc. In the network several traffic flows (F_i) may be active at the same time. Each node can be identified using (x, y) coordinates. The router located at coordinates (x, y) is referred to as $R(x, y)$.

Routing decides the path that a packet follows within the network, and consequently, the number of routers or *hops* (h), a given flow requires to move from a source to a destination node. Hence, a router can also be identified as R_i^j, in which j represents the hop j of flow F_i, when moving to its destination.



(a) (b)

Fig. 2. Mesh basics. (a) Router coordinates in a 4x4 part of a mesh. (b) Canonical 2D-mesh router.

Communication flows comprise multiple NoC packets or requests. We refer to the k-th packet generated by flow F_i as r_i^k. A packet is the minimum arbitration unit in the network and it can be split into one or several *flits* (short of control flow units). Flits can be further decomposed into smaller units called phits when available link wires cannot accommodate an entire flit. For the sake of clarity in the equations we consider equal phit and flit sizes. The first flit of a packet is called header flit and contains the information required to forward the packet to the destination.

We consider a canonical 5-port[2] 2D mesh router architecture in which input ports have a queue to store packet flits (see Figure 2(b)). In order to alleviate the contention caused by flows going to different destinations, high performance NoCs multiplex physical channels using *virtual channels* (VC). To do so, an input queue resource per port is assigned to a virtual channel. In a canonical wormhole router with virtual channels, two rounds of arbitration are performed. The first selects the input port that is granted access to a given output port and the second one selects the virtual channel (queue) that is selected in a given input port. In the case of a router without virtual channels the latter arbitration is not required.

Routers are usually pipelined in multiple stages, e.g. the Intel SCC [30] comprises routers with an input buffer, routing of the header flit, switch allocation and link traversal stages. The header flit is the only one arbitrated from a packet and once it is granted access to a given output port this connection remains established until the entire packet leaves the router.

When a header flit arrives at an input port of R_i^j, this flit is stored in the corresponding queue. Next, the routing determines the next hop router, R_i^{j+1}, and the router allocates an entry queue in R_i^{j+1}. Once the router in the next hop can accept the header flit, it competes for an output port and traverses the router crossbar. Once a header flit is granted access to a given output port, the remaining flits of the packet are forwarded to this port without any further arbitration.

However, contention may cause the header flit to be stalled. When this happens, the remaining flits of the packet are also stalled. One of the causes of stalls is the finite size of queues in input ports. In wNoCs, the minimum allowable queue size is one flit. In any case, queues are typically sized with enough space to avoid bubbles in the packet transmission. For instance, if the time required to know if there is enough space in the next router queue is equal to one cycle, the queue needs to have a minimum size of two flits to avoid bubbles. The latency experienced by a packet to traverse the network from source to destination in the absence of contention is usually referred to as *zero-load latency* (zll).

[2]In order to simplify our formulation we assume that all routers, including those at the edges, have the same number of ports, which in our case is 5. We could consider that some routers (e.g. those at the edges) may have fewer ports, which would decrease the WCD. However, for the sake of clarity and due to space limitations to extend equations we stick to 5-port routers for meshes as the ones used in Tilera chip [36].

TABLE II
LIST OF WNOC MAIN FEATURES ANALYZED

ID	Feature	Comment
-	Routing	Fixed to achieve time analyzability. Static (e.g. XY routing)
-	Flow control	No impact. Credit-based or stall-and-go.
-	Arbitration	Fixed to achieve time analyzability
-	Switching	Fixed to wormhole (widely implemented).
cF	Number of flows	Limited by static routing.
nVC	No. of queues per input port	$nVC = 1$ or $(1 < nVC < cF)$
E	Entries per queue	< 1 packet, $= 1$ packet or > 1 packet
S	Packet size	Single or Multiple
FT	Flits per Packet	$= 1$ or > 1

B. Proposed Taxonomy

The main properties the wNoC needs to provide in order to be used in real-time systems are (i) time analyzability, i.e. enabling the derivation of as tight as possible contention delay bounds, and (ii) time composability, i.e. making contention delay bounds independent of the load that co-runners put on the wNoC. This translates into deriving the trustworthy upper-bound to the highest possible contention delay ($\overline{WCD_i}$) a communication flow F_i of a given task can suffer due to conflicts with other task's flows. In the following we show how different wNoC parameters impact $\overline{WCD_i}$. Table II summarizes the wNoC features we analyze.

Fixed parameters. Some wNoC parameters are usually constrained to enable time analyzability and composability:

Routing determines the flows that potentially contend with F_i at a given router. Deterministic routing is shown to provide time analyzability [29]. Hence, for our mesh analysis we use *XY* as it is the preferred solution for routing in regular NoCs due to its low implementation cost. With XY routing packets are forced to use the X dimension first: In the X dimension the position of the target node with respect to the source node determines whether to go right (X+) or left (X-) direction. The same approach is used for the Y dimension. Once a packet is routed using the Y dimension, it cannot be forwarded back to the X dimension. These routing restrictions determine the maximum number of flows contending with F_i at a given router for an output port. Note that the opposite port of a given input/output port is represented as $\bar{Y}$ and $\bar{X}$.

Note that our analysis can be extended for any other deterministic routing policy. In order to do so, one should

recompute the maximum number of flows contending with F_i at a given router for an output port according to that particular routing policy.

Flow control determines how packets traverse the routers. In the context of wormhole switching, back pressure flow control can be based either on the use of credits or stall-&-go signals. In this paper we provide expressions assuming the most common case that the flow control mechanism is designed in such a way that no bubbles occur in the packet transmission. However, the impact of bubbles on contention delay can be easily accounted for by considering that a bubble in the transmission is equivalent to increasing packet size by one flit.

Active nodes. In order to achieve time composable contention delay bounds, no assumptions can be made on the particular active flows in the wNoC. That is, it is assumed that any node in the network is entitled to send and receive packets from any other node.

Active flows. Similarly, when computing the contention delay for a packet, we assume that, by the time it is injected in the network, any other potential contending flow is also active at that moment, transmitting its packets in a way that it produces the worst possible contention scenario. In order to reproduce the worst possible contention scenario we need to consider the *worst direct contention* and the *worst indirect contention* [17]. The former can be easily reproduced by considering that for a packet r_i^k of F_i at every hop, all possible contenders (i.e. all queues that can forward a packet to the requested output port) are also requesting the same output port. The latter is caused by packets of flows not sharing the path with F_i but blocking at least one flow that does share at least one link in the path with F_i. In the following sections we derive expressions that account for worst-case contention considering the impact of both indirect and direct contention.

Output port arbitration. Likewise, packets contending in a router for a given output port are arbitrated using a time-analyzable policy. Regular wormhole-based mesh designs like the ones in [36] [30] use round-robin arbitration. The use of round-robin arbitration enables the computation of timing guarantees [15] [24].

Parameters to adjust. Other wNoC parameters have some flexibility in the values they can take, though each set of parameters (network parameter configuration) leads to different contention delay. We study the following set of parameters: buffer capabilities and number of flits per packet.

The *buffering capabilities* of the wNoC are further shaped by two parameters:

• The number of queues per input port – which matches the number of virtual channels – (nVC). nVC leads to two scenarios: First, when there is a single queue per input port, i.e. no virtual channel is implemented, which is referred to as $nVC = 1$; second, there are several queues each of which can – statically or dynamically – hold different flows ($1 < nVC < cF$).

• The number of entries per queue (E) that can get three values: it can have the exact size of a packet, given by its number of flits ($E = 1$), be smaller than packet size ($E < 1$) and be larger than packet size ($E > 1$).

For the number of *flits per packet* we consider two cases: Each packet comprises a single flit ($FT = 1$) and each packet comprises several flits ($FT > 1$).

TABLE III
SETUPS

	Setup	Description
Impact of VC	$(FT = 1, nVC = 1, E = 1)$	1 queue per input port, 1-flit packets and input queue holds 1 packet
	$(FT = 1, 1 < nVC < cF, E = 1)$	nVC queues per input port, 1-flit packets and input queue holds 1 packet
Impact of FT and E	$(FT > 1, 1 < nVC < cF, E = 1)$	Input queue holds 1 packet that is multi flit
	$(FT > 1, 1 < nVC < cF, E > 1)$	Input queue holds more more than 1 packet that is multi flit
	$(FT > 1, 1 < nVC < cF, E < 1)$	Input queues cannot store entire an entire packet that is multi flit

V. TIME-COMPOSABLE WCD BOUNDS

This section provides an analytical model for time-composable bounds to $\overline{WCD}$ with the ultimate goal of deriving time-composable bounds to tasks execution time. Table III presents the different scenarios we analyze in coming sections. In doing so, we proceed incrementally analyzing the contention delay affecting packets in the NoC, going from simple scenarios to more complex and realistic ones.

A. Single-Flit, One Virtual-Channel, Single-entry Queue ($FT = 1, nVC = 1, E = 1$)

In this scenario, packets are composed of one single flit and every router input port has a single-entry queue (no virtual channel is implemented). The queue stores the requests coming from all flows sharing it.

Single-router traversal. Let us assume a request r_i^1 from a flow F_i going from a source node R_i^1 to a destination node R_i^2 that are adjacent in the direction of F_i. In this wNoC setup, traversing one router is similar to traversing a bus with round robin arbitration policy [15]. The worst contention delay that r_i^1 can experience is:

$$\overline{WCD_i} = (NR_i^1 - 1) \times L_{router} \quad (1)$$

NR_i^j is the number of queues contending for an output port that F_i is targeting at router R_i^j. With XY routing if the destination port is $X+$ or $X-$, the number of contending queues is 2 (PME and $\bar{X}$). If the destination port is $Y+$ or $Y-$ (or the PME) the contending queues are 4: $X+, X-, \bar{Y}$ and PME (or $X+, X-, Y+$ and $Y-$). L_{router} represents the time a packet requires to cross a non-pipelined router. In the case of pipelined routers, the pipeline mitigates the impact of L_{router} and it can be safely assumed $L_{router} = 1$. In the rest of the paper, we make this assumption for the sake of clarity and readability.

Worst Contention. Contention is caused by packets of any flow partially sharing the path with F_i. Let's assume

F_i traverses R_i^1-R_i^2-R_i^3. When r_i^1 is issued from the PME it enters the arbitration in R_i^1. In the highest contention scenario $NR_i^1 - 1$ requests with higher priority than r_i^1 are ready in R_i^1 per-input-port queue targeting the same output port. For any of these requests to go through R_i^1 the corresponding input queue in R_i^2 input port has to be free. In the worst-case situation, the target input port already contains a packet (r_j^k) from a different flow, F_j, and r_j^k shares the same path as r_i^1. Further, all packets in different input ports in R_i^2 can target the same output port as r_j^k and have higher priority than r_j^k. Overall, we observe that r_j^k causes contention on r_i^1 despite not contending within the router R_i^1 as they share at least one link in the path (direct contention). Additionally, requests not sharing the path with r_i^1 can be blocking r_j^k which in turns causes contention in r_1^1. This contention is usually regarded as indirect contention [17].

Worst-Possible Destination. From the previous discussion it follows that the route followed by r_j^k determines the contention it suffers, so the more hops r_j^k traverses the more the contention it may suffer, which in turns affects the contention on r_i^1. In order to account for the worst contention, considering both indirect and direct contention that any F_i can suffer at hop R_i^j, we introduce the concept of *worst-possible destination flow*, $F_{\omega(i,j)}$. $F_{\omega(i,j)}$ considers the next hop's input port that a packet of flow F_i targets from current hop R_i^j. The destination of flow $F_{\omega(i,j)}$ is chosen in such a way that causes worst indirect contention to the packets of flow F_i, i.e. it prevents packets of F_i cross the hop R_i^j for the longest time possible.

The choice of $F_{\omega(i,j)}$ depends on the routing algorithm used. In the wNoC mesh considered in this paper, with XY routing, the worst destination of flow $F_{\omega(i,j)}$ corresponds to the farthest node that can be reached from the next F_i hop's input port[3], depending on the traversing direction.

- When the packets of flow F_i traverse the Y dimension, the farthest reachable node is the one with highest hop count in the same direction, since once a packet starts using the Y direction, it cannot be routed on the X direction again.
- When the packets of flow F_i traverse the X dimension, the farthest node is the one with highest hop count in X and then in Y dimension.

Let's consider the case from Figure 3(a) in which a packet r_i of flow F_i is transmitted from router $R(0,1)$ to router $R(3,0)$ (represented with a solid arrow in the figure). When the packet enters in R_i^1, it waits for an input port to become ready in R_i^2. r_i^1 suffers the longest contention when the packets in the west input port of R_i^2 target their worst possible destination, i.e router $R(3,3)$ (represented with a flow $F_{\omega(i,1)}$, dotted arrow in the figure). The same worst possible destination is maintained as the packet traverses $R(1,1)$ and $R(2,1)$. However, when r_i reaches router $R(3,1)$ as the packet requests the north input port in router $R(3,0)$, the worst possible destination changes. The reason is because $F_{\omega(i,1)}$ considers Y+ but r_i goes Y-. As a result, a new worst possible destination flow is computed, i.e. $F_{\omega(i,4)}$, marked with a dashed arrow in Figure 3(a).

[3]This assumption is only valid in the case all routers have the same number of ports. In other cases the worst possible destination is computed iterating contending flows to the possible destination and selecting the one causing the highest contention.

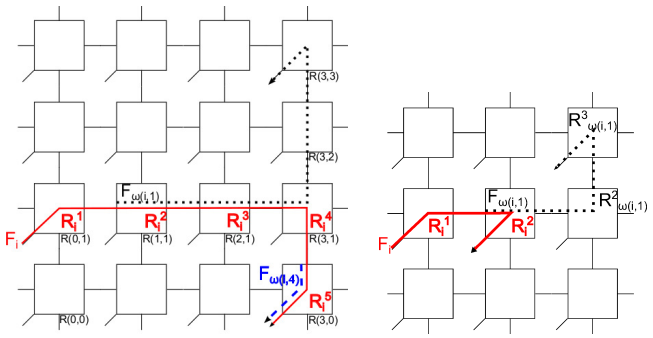


Fig. 3. (a) Worst destination; and (b) A flow crossing 2 routers

Computing the time-composable upper bound Worst-Contention Delay ($\overline{WCD_i}$). In order to derive the general $\overline{WCD_i}$ expression, we first focus on the case of a flow crossing only two routers as shown in Figure 3(b). Flow F_i targets next hop's (R_i^2) PME. In order to cross router R_i^1, it has to win the arbitration for the east output port of R_i^1. To keep time composability we assume that it has the lowest priority in that arbitration, as shown in Equation 1. We further assume that each of its contenders in R_i^1 suffers the worst contention from $F_{\omega(i,1)}$ (marked with a dotted arrow in the figure), which determines the delay suffered by each contender in the arbitration.

In order to compute the impact that $F_{\omega(i,1)}$ has on contenders of F_i, we follow an iterative process, assuming that $F_{\omega(i,1)}$ also suffers the worst contention at each hop until reaching its destination $R_{\omega(i,1)}^3$. Thus, the worst contention for every request going from $R_{\omega(i,1)}^2$ to $R_{\omega(i,1)}^3$ is $NR_{\omega(i,1)}^3$. Likewise, the contention when going from $R_{\omega(i,1)}^1$ to $R_{\omega(i,1)}^2$, is $NR_{\omega(i,1)}^3 \times NR_{\omega(i,1)}^2$ as it includes the contention of $R_{\omega(i,1)}^2$ when going to $R_{\omega(i,1)}^3$.

Overall, the worst contention that $F_{\omega(i,1)}$ causes on F_i contenders at router R_i^1, includes the arbitration of $F_{\omega(i,1)}$ at router $R_{\omega(i,1)}^1$ and is equal to $NR_{\omega(i,1)}^3 \times NR_{\omega(i,1)}^2 \times NR_{\omega(i,1)}^1$. As in the previous case, it includes the contention of $R_{\omega(i,1)}^1$ when going to $R_{\omega(i,1)}^2$ and $R_{\omega(i,1)}^3$. Once the impact of $F_{\omega(i,1)}$ is computed, we derive from Equation 1 the contention that F_i suffers for crossing from R_i^1 to R_i^2. As a result, the $\overline{WCD_i}$ expression when crossing two routers is:

$$\overline{WCD_i} = NR_{\omega(i,1)}^3 \times NR_{\omega(i,1)}^2 \times NR_{\omega(i,1)}^1 \times (NR_i^1 - 1) + (NR_i^2 - 1) \quad (2)$$

In the general case, for a packet of an arbitrary flow F_i injected in an arbitrary node R_i^1 and that it has to cross H_i other routers (R_1, R_2, ... R_{H_i}) to get to its destination, the general expression for computing $\overline{WCD_i}$ of F_i is given by:

$$\overline{WCD_i} = \sum_{j=1}^{H_i} \left((NR_i^j - 1) \times \prod_{m=1}^{H_{\omega(i,j)}} NR_{\omega(i,j)}^m \right) \quad (3)$$

where $H_{\omega(i,j)}$ is the number of hops in the worst possible destination flow $F_{\omega(i,j)}$. The first multiplicand $(NR_i^j - 1)$ corresponds to the *contention introduced by the round robin arbitration* in each of the routers that the flow F_i traverses. The second multiplicand $\prod_{m=1}^{H_{\omega(i,j)}} NR_{\omega(i,j)}^m$ corresponds to the *indirect contention delay* in each hop due to the worst possible destination flow $F_{\omega(i,j)}$. In the rest of the paper, we refer to the first multiplicand as rr_{cont}, and to the second as ind_{cont},

Interestingly, whether or not a node has several requests in flight has no impact on $\overline{WCD}$, since this only affects *atd* and *WCD* metric is insensitive to *atd*.

B. Single-Flit, Virtual-Channels, Single-entry Queue ($FT = 1, 1 < nVC < cF, E = 1$)

Virtual channels are allocated to flows in a wNoC either statically or dynamically. With dynamic allocation, virtual channels are assigned to flows at run-time based on their availability with the overall goal of maximizing buffer occupation and consequently, average network throughput. With static allocation instead, virtual channels are statically assigned to a given set of flows, such that any of these flows uses the same virtual channel until reaching the destination, in order to reduce head-of-line blocking [9]. The way in which each of the two allocation schemes impacts wNoC contention is as follows:

For dynamic allocation, at analysis time no assumption can be made about the particular flows contending at a given router with F_i. The only safe assumption that can be made is that all flows can be potentially contending for virtual channel resources at every router. Hence, dynamic virtual channel allocation does not help reducing contention delay.

The impact of static virtual channel allocation is more complex to ascertain. Hence, if we consider terms rr_{cont} and ind_{cont} from Equation 3:

• An increase in nVC translates into a increase of rr_{cont}. This occurs because every arbitration round covers the selection of a contending port (e.g. for the Y+ output port the contending input ports are $X+$, $X-$, $\overline{Y}$ and PME) and a specific virtual channel of that port. Hence, the number of contenders within a router, NR_i^j, in the presence of nVC virtual channels per input port, is defined depending on the destination port as follows:

$$NR_i^j = \begin{cases} nVC \times 2 & if\ destination\ is\ X+\ or\ X- \\ nVC \times 4 & if\ destination\ is\ Y+,\ Y-\ or\ PME \end{cases}$$

Note that the expression above generalizes the definition of NR_i^j presented in Equation 3, which considers no virtual channels are implemented ($nVC = 1$).

• Having more than one virtual channel, if they are smartly allocated, offers a solution to reduce ind_{cont}. The achieved reduction factor, referred to as Δ_{ind} (with $\Delta_{ind} \leq 1$), depends on the particular static allocation used. For instance, let us assume a wNoC with two virtual channels, one of which is assigned to packets sent from a given node $R(x, y)$ while the other nodes share the second virtual channel. In this scenario the packets from $R(x, y)$ suffer no indirect contention, i.e. $rr_{cont} \times \Delta_{ind} = 1$, hence reducing $\overline{WCD}$ since the reduction is expected to be higher than the increase caused on rr_{cont}. It is noted that the requests sent from the other nodes suffer a higher $\overline{WCD}$ since their ind_{cont} is not affected while rr_{cont} increases. Investigating smart static allocation virtual channel policies is out of the scope of this paper and remains a part of our future work, in terms of the $\overline{WCD}$ model presented in this paper we use the terms NR_i^j and Δ_{ind} to factor in these effects into $\overline{WCD}$. Overall, the general expression for $\overline{WCD_i}$ is as follows:

$$\overline{vcWCD_i} = \overline{WCD_i} \times nVC \times \Delta_{ind} \quad (4)$$

C. Multiple-Flit, Virtual-Channels, Single-entry Queue ($FT > 1, 1 < nVC < cF, E = 1$)

In this section we model wNoC contention when packets can have more than one flit, which are transmitted in a pipelined fashion. In order to account for the contention delay introduced by multi-flit packets, we consider L_i^{flit} as the maximum number of flits a packet of flow F_i can have.

In a pipelined router, the time that a packet r_i^k is blocked in R_i^j is given by the number of queues that can potentially contend with r_i^k ($NR_i^j - 1$) and the maximum number of flits (L_i^{flit}) of the contending requests: $(NR_i^j - 1) \times L_i^{flit}$. As a result, in order to provide a contention delay bound ($\overline{WCD_i}$), it is required to know the maximum packet length of every flow in the wNoC as in [7]. However, the latter breaks time-composability. In order to retain time composable behavior while supporting any bounded length packets, we define L_{MAX}^{flit} as the maximum length that packets in the wNoC can have. In this context, the general expression for $\overline{ftvcWCD_i}$ can be formulated as follows, based on Equation 3:

$$\overline{ftvcWCD_i} = L_{MAX}^{flit} \times \overline{vcWCD_i} \quad (5)$$

Note that typically L_{MAX}^{flit} is determined by the communication protocol (e.g. [1]) on top of the wNoC. Also, it can be limited at network interfaces by performing packetization [31].

D. Multiple-Flit, Virtual-Channels, Multiple-entry Queue ($FT > 1, 1 < nVC < cF, E > 1$ or $E < 1$)

For $\overline{WCD}$ Equations so far, we have considered that queue size is equal to packet size. In this section, we consider the impact of having queues not matching packet size.

Queue size larger than packet size ($E > 1$). When queues have enough space to store more than one packet, the number of in-flight packets in the network increases. However, this affects the worst-case latency experienced by packets in the wNoC but not its contention delay. This is because packets in a given virtual channel queue are served using a *first-in*

first out policy and therefore fairly arbitrated by round-robin arbiters across router ports. In fact, the maximum number of packets that r_i^k is contending with at a given router R_i^j is given by NR_i^j and is the same regardless the number of contending packets that can be stalled in a given router port. In this case Equation 5 is a valid expression for this case ($\overline{ftvcWCD_i}$).

Queue size smaller than packet size ($E < 1$). When a packet cannot be completely buffered in a router, its flits are spread across several of the router queues the packet traverses until reaching its destination node. The number of effective contenders in a network with buffers smaller than packet size is reduced since a given packet cannot be requesting an output port at two routers at the same time [29]. With this in mind it can be concluded that the resultant $\overline{WCD}$ is equal or smaller than the one derived in Equation 5 ($\overline{ftvcWCD_i}$). However, composability requirements make almost impossible determining the number of effective contenders as this would require knowing the exact length for all packets of all flows in the network. Therefore, in this scenario a safe upper bound is $\overline{ftvcWCD_i}$.

VI. System design considerations

Incremental verification calls for composable WCET estimates that are not affected by other applications. At the NoC level this translates into each request to account for a time-composable upperbound to the contention delay ($\overline{WCD}$) it can suffer. From Equation 5 it follows that worst-contention delay depends on three main factors: (1) the highest packet size a flow may have (L_{MAX}^{flit}), (2) the number of VC (nVC), and (3) the network size. This section discusses about these three effects when designing a critical real-time embedded system and reviews some existing techniques that can be employed to minimize their impact.

Packet Size. Interestingly some NoC designs limit the maximum packet size by hardware and on others this responsibility is left to the software layer. In the former case, the hardware enables L_{MAX}^{flit} to be factored in WCD, which in turn enables bounding the impact that other flows on the WCD.

However, in the latter case, a low priority task may send long (or even unbounded) packets over the network, thus increasing – potentially in a unbounded manner – the WCD of high-priority tasks. In this context, it is required a suitable mechanism allowing high-priority tasks to be isolated from flows having unbounded packet size.

Virtual Channels. The use of virtual channels, which need to be allocated in a static manner, helps reducing $\overline{WCD}$ under the conditions presented in Section V-B. Interestingly, in a mixed-criticality system, allocating each criticality level an independent VC does not help reducing WCD. First, VCs increase rr_{cont} since it is multiplied by nVC. Further, if no constraint is put on the number of cores that in a given point in time can send requests under a criticality level, ind_{cont} is not reduced, i.e. $\Delta_{ind} = 1$. Moreover, L_{MAX}^{flit}, which captures the impact that the longest packet transmitted by any flow causes on WCD, is independent of the particular VC in which that packet is allocated (see Section V-C).

The dual-criticality systems, for instance, in the space domain, it is well accepted that on-board systems comprise two criticality levels [25]: one for *control* applications requiring real-time execution, and another for *payload* applications that are high-performance driven and have some (soft) real-time requirements. In such dual-criticality systems having one virtual-channel per criticality level may help reducing the WCD suffered by requests of high-criticality tasks due to low-criticality ones. This requires prioritizing high-criticality requests over low-criticality ones. Flit-level preemption can also be used to further reduce this impact. However, this comes at the cost of providing no WCD guarantees to the low-criticality tasks', since their requests' WCD depends on the load high-criticality tasks put on the NoC. In other domains, such as avionics or automotive, comprising more than two criticality levels, with several of them requiring time guarantees, the dual-criticality approach does not apply. Investigating static VC allocation policies for these domains is a fertile area of research and part of our future work.

Network Size. WCD directly depends on the mesh size. Clustered designs like those proposed in [23] allow creating independent islands of communication (i.e. clusters) within the wNoC by properly routing packets, which in turn reduces the WCD. However, in this case a mechanism is needed to allow inter-cluster communication without jeopardizing the WCD of the affected clusters if they are intended to run real-time tasks. In [23] inter-cluster communication is allowed by using multiple VCs. This approach proposes the use of two VC: one for intra-cluster and another one for inter-cluster communication assuming the amount of inter-cluster communication is a known parameter at system integration. While this holds for the case of communication requirements defined in avionics applications [4], this would jeopardize time-composability in other domain applications. partition communications can be isolated without jeopardizing time-composability.

VII. Modeling existing NoC designs

In this section we first assess the accuracy of the WCD model presented in Section IV with special emphasis on the accuracy of $\overline{WCD}$ in the case of the most complex wNoC designs ($\overline{ftvcWCD_i}$). We also compare WCET estimates obtained when both WCD and WCTT are used to capture the impact of the wNoC contention on application's WCET. Finally, we evaluate the impact that different network parameters have on the contention.

Our experimental setup comprises *gNoCsim* [2], a powerful cycle-accurate simulator of wormhole networks developed in the context of the NaNoC project, which we connect to an enhanced version of the *SoClib* simulator [3] to model a complete manycore processor. With this framework, we model two network setups resembling the ones deployed in real processors: the TileraGx36 [36] and the 48-core Intel SCC (ISCC) [30] manycore designs, based on publicly available data. Table IV shows the relevant parameters of the two different wNoC setups. The main difference between these two network setups is on the usage of virtual channels. The

TABLE IV
TECHNICAL DETAILS OF THE MESH NOC IN HIGH-PERFORMANCE CHIPS: 48-CORE INTEL SCC AND 36 CORE TILERA-GX36

	size	routing	L_{router}	nVC	wNoCs	Link width	L_{flits}^{max}
Intel SCC	6x4	XY	4 cyc	8	1	128 bit	4
Tilera-Gx36	6x6	XY	1 cyc	1	5	32 bit	16

ISCC implements a wNoC with eight virtual channels and the Tilera-Gx36 chip uses five independent networks that are used to completely isolate different types of traffic.

A. $\overline{WCD}$ accuracy and comparison with WCTT

We assess the accuracy of our $\overline{WCD}$ model in upperbounding the actual contention caused in the wNoC by creating a high-congestion scenario. To that end, we simulate the traffic generated by a memory-intensive scenario in which all cores in the network send packets to memory continuously. Note that such traffic can be produced in real scenarios by programs writing to memory continuously. In fact, we have noticed that similar congestion scenarios can be reproduced even when each node only sends packets sporadically if they share the same destination node.

In this experiment, for all the network setups we analytically compute bounds to WCTT [29] and $\overline{WCD}$ (Equation 5) and compare them with the measured contention delay and worst-case traversal time obtained with the simulator under highest congestion scenario. It is noteworthy to mention that the model in [29] computes bounds provided the existing flows in the system at deployment time are known and thus, precluding incremental verification. To enable a fair comparison of our metric (WCD) with the WCTT metric we have adapted the expressions in [29] to achieve composable WCTT bounds by considering an all-to-all communication scenario. Note that it would also be possible to adapt WCD expressions to compute bounds for a particular application. However, in this paper we study the benefits of WCD over $WCTT$ in a time-composable scenario. In the modeled high-congestion scenario, the measured contention delay and traversal time closely match actual WCD and WCTT respectively. To ensure a steady congestion state is reached, measurements do not start until at least 1,000 packets per node have been injected and are performed until all nodes have sent at least 2 million requests.

Figure 4 shows a comparison of the measured and computed metrics in both Tilera and ISCC-like networks. We assume that buffers have capacity to store two packets to avoid bubbles in the network. The results in each bar show the geometric mean (gmean) of WCD (and WCTT) for all the communication flows in the wNoC (i.e. for the packets of all nodes). This provides a measure of the WCD (WCTT) each packet suffers on average. All values are normalized to the measured (observed) contention delay.

- The derived $\overline{WCD}$ bounds are always higher than the measured contention delays confirming they upperbound the contention in the wNoC. Moreover, the difference between measured and predicted contention with our model is very small: 5% on average and 7% in the worst case.
- Likewise, measured WCTT values are close to the predicted ones [29]. The difference between WCD and WCTT (which is roughly the same for measured and predicted values) is significant, evidencing that WCTT can be a pessimistic metric to account for the interference of co-running tasks in the network. In particular, for the ISCC WCTT is 2.94x higher than the $\overline{WCD}$ and 2.7x higher for the Tilera.

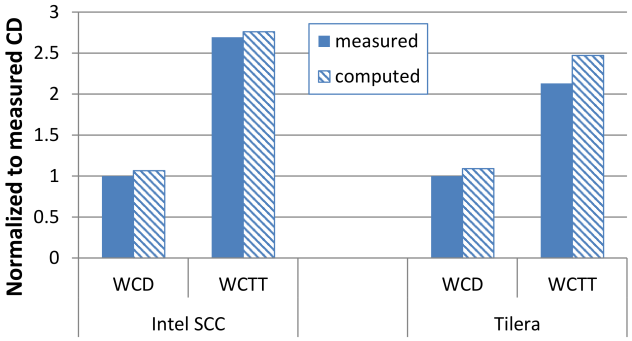


Fig. 4. WCD bounds derived in this paper and adapted WCTT from [29]

It is worth mentioning, although it is not presented in any chart, that we have observed that, unlike $\overline{WCD}$, WCTT grows with buffer size which in turns makes this metric even more pessimistic when using wNoCs with larger buffer sizes.

B. Reducing $\overline{WCD}$ values

Another parameter with high impact in wNoC contention is the number of VCs and how they are allocated. In Figure 5 (in logarithmic scale) we show the effect of reducing in the ISCC-like wNoC the number of VCs from 8 to 1. We observe a reduction in terms of WCD (both observed and predicted) of more than 7 times. Further, a smart deployment of the wNoC by, for example, using regions of execution (clusters) as presented in Section VI and properly mapping applications to cores [23] produces reduced contention delays. If we further create clusters of size $3x4$ or $3x2$ contention delay is additionally reduced. Note that all those adjustments can be done from the software without any change at hardware level in the wNoC design. For instance, islands can be implemented by mapping application so that communication doesn't not exceed defined island [23]. Likewise, the number of VCs is a parameter that can be easily changed from software. Researching on the convenient wNoC configurations (regions, static VC allocation) is part of our future work, building on the contention delay model developed in this paper. Finally, it is worth noting that in all scenarios in Figure 4, our $\overline{WCD}$ model tightly captures the impact of these parameter variations.

C. Impact of wNoC interference on WCET

For the experiments in this section we use EEMBC Autobench benchmarks [26]. We execute benchmarks in 3 different scenarios: *2-hop* where the memory is 2 hops away (1 in X and 1 in Y dimension) from the core where the benchmark runs;

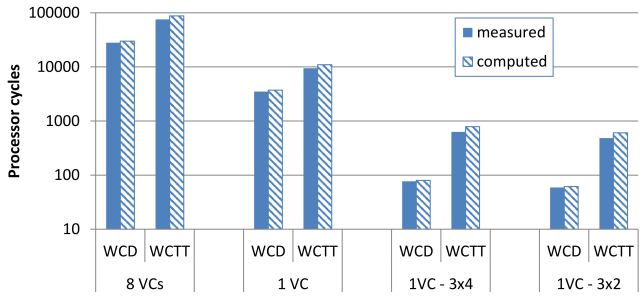


Fig. 5. Effect of disabling VC and clustering on WCD for the SCC setup

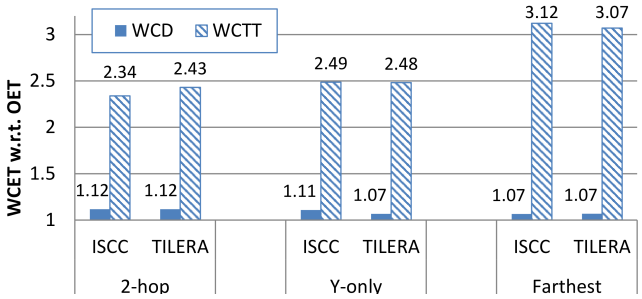


Fig. 6. $WCET_{mc}$ estimates derived with WCD and WCTT w.r.t. OET

Y-only where the benchmark is executed on the node farthest away from the memory in the Y-axis and *farthest* in which the benchmark is placed most hops away from the memory.

We compare observed execution times (OET) against WCET estimates. The former is computed, by running the application under a high-congestion scenario (as explained in previous section) in order to provide fair comparison to time-composable WCET estimates. For the latter, we first compute the worst case execution time of the application using zero-load latency ($WCET_{zll}$) and increment it with the predicted impact of the wNoC (Δ_{wNoC}). As a result, the WCET estimate for manycore ($WCET_{mc}$) is computed as follows:

$$\begin{aligned} WCET_{mc} &= WCET_{zll} + \Delta_{wNoC} \\ \Delta_{wNoC} &= n_{req} \times \begin{cases} \overline{ftvcWCD_i} \\ WCTT - zll \end{cases} \end{aligned} \quad (6)$$

where n_{req} is the number of requests the application makes to the wNoC along the worst-case path. Note that WCTT already includes zero-load latency and in order to provide a fair comparison, we have to deduct zll from WCTT, as it is already included in $WCET_{zll}$.

Figure 6 shows the gmean of $WCET_{mc}$ estimates of EEMBC benchmarks obtained with both $\overline{WCD}$ and WCTT. We observe that $WCET_{mc}$ estimates obtained with $\overline{WCD}$ are between 7% and 12% higher than OET. The maximum difference for any benchmark is around 16% in the 2-hop scenario. Meanwhile, in the case of $WCET_{mc}$ estimates obtained with WCTT there is a significant difference w.r.t. OET. They are between 2.3x and 3.1x higher, with a maximum difference of 3.2x across benchmarks.

VIII. Related Work

Several network designs targeting soft and hard real-time systems have been proposed. Next we summarize them.

Quality of Service (QoS). In the high-performance and high-end embedded domain, QoS is used as a metric to measure time predictability. Several proposals exist to improve predictability on wNoCs, e.g. [10]. These QoS techniques are specially suitable, for multimedia applications since QoS can be offered under severe network load conditions. Unfortunately, these techniques make difficult deriving tight contention-delay bounds to each request to the wNoC, which challenges deriving guarantees on that tasks running in a wormhole-based manycore will meet their deadlines. Authors in [6] proposed the QNoC architecture, which offers several degrees of guarantee at low hardware cost. However, despite that achieving real-time traffic guarantees is one of the targeted services in QNoC, latency bounds provided in this study do not actually bound contention delay experienced in the wNoC, preventing the derivation of time-composable WCET bounds.

Real-time Specific NoCs. While there are several proposal of real-time aware NoC designs – some of which have implemented in FPGA or implemented in real products (e.g. in the multimedia domain) –, exploring to which extend high-performance (COTS) NoC designs can be used in the real-time domain is of paramount importance. It is well accepted that the hard real-time domain is a relative small market in comparison with other domains such as mobile. Hence, customized NoCs tailored for real-time such as TDMA-based or time-triggered ones will naturally find difficulties in being adopted by real-time industry [35] since their implementation incur high non-recurrent costs. On the other hand, the big majority of the proposed manycore designs across all computing domains use high-performance wNoCs to perform the interconnection of cores and shared resources within the chip. This makes wNoCs accessible (at low cost) by the CRTES as they are implemented in a vast set of chips. This paper makes an effort in the direction of understanding the limitations and challenges in adoption of wNoCs in real time systems.

Although it is not the topic of this paper in the field of real-time specific NoCs we highlight TDMA-based NoCs [13], [14], [32] approaches that satisfactorily provide time composable behavior. While this TDMA-based NoCs that deal with contention at transaction level (e.g. read and write memory operations), time-triggered architectures [22] increase the abstraction level by introducing the notion of a micro-component, which is a self-contained computational unit. In time-triggered architectures micro-components exchange messages in contention-free slots. However, event-triggered transactions, such as cache misses that access main memory through the NoC, may suffer contention delay which also must be upper-bounded.

Several hard real-time wormhole-based designs use of flit-level virtual-channel preemption mechanisms for dual-

criticality systems [21], [33] to control contention in the network in order to reduce network latency. In these approaches high-priority virtual-channels can preempt packets from low-priority virtual channels so that contention delay high-priority channels is reduced at the cost of removing time-composable contention delay guarantees to low-priority channels. They require a virtual channel per communication flow, which limits their scalability. This limitation is addressed in [34] by proposing a priority share policy where contending flows can share a given priority. Along this line, a recent work [7] proposes an enhanced priority-shared flit-level virtual-channel preemption NoC to support two criticality operation modes. This design fulfills isolation requirements across criticality levels without incurring in hardware resource wasting. Further, in [16], authors build on top of [33], [34] and provide response time analysis which considers impact of pipelining in the NoC. However, this analysis considers communication among tasks and do not consider memory accesses inside a task.

In general, flit-level virtual-channel preemption mechanisms offer tight contention delay estimates. However, these approaches consider the impact of contention delay at the schedulability and response time analysis and require knowing the exact set of tasks using the wNoC and their communication flows, which negatively affects time composability and incremental verification, as discussed in Section II.

WCTT in wNoCs. Several works focus on deriving an upperbound of WCTT adapting network calculus [18] to model wNoCs [28]. Network calculus relies on the determination of arrival curves of the applications running in the system to determine an actual upperbound of WCTT. While these approaches allow providing tight WCTT estimates, as WCTT is adapted to the exact network load conditions, using per-application arrival curves reduces time composability, since WCTT estimates depend on the load corrunners put on the NoC. Another set of works focuses on determining wNoC packets WCTT by considering worst-case conditions, first with assuming limitations on the packet-injection rate [19]. However, the bounds provided in [19] required assuming packet injection is limited. In later works [11], [29] this limitation is removed.

In this line of work, Dasari et. al [8] achieve tighter WCTT bounds – than those derived with [11], [29] – based on the following two observations: (1) flows injection rate is inherently limited by the speed at which the processor pipeline can process request-generating instructions (e.g load or stores); and (2) packets of a given flow do not always contend with the flow under analysis due to the existing release time of their request of a flow/core. Regarding observation (1) we have shown that while limiting the injection effectively reduces WCTT values the actual contention that packets in the NoC suffer remains unaltered, hence producing no effect on WCD. Regarding observation (2) knowing what is the actual interval between consecutive requests in every flow in the network breaks the time composability requirement.

IX. Conclusions

In this paper we analyze the suitability of applying wNoCs in the real-time embedded domain. To do so, this paper proposes a new metric to account for the impact that NoC interferences coming from different requesters have on the WCET estimates: the *worst-case contention delay* (WCD), which replaces the traditional metric, the *worst-case travel time* (WCTT). Moreover, we derive an analytical model that computes time-composable WCD bounds ($\overline{WCD}$) based on common wNoC design parameters including flits-per-packet, number of virtual channels and queue size in the router. $\overline{WCD}$ is computed based on a wNoC parameter taxonomy that identifies those parameters that must be fixed in order to provide trustworthy and composable WCD bounds; and those allowing certain flexibility.

Our $\overline{WCD}$ model allows evaluating a wide range of existing COTS high-performance wNoCs. To that end, we apply the model considering the design parameters of two wNoCs deployed in real processors: the Tilera-Gx36 and the 48-core Intel SCC (ISCC). Our analysis shows that considering WCD rather than WCTT reduces WCET estimates by around 2.5x for Tilera and ISCC on average.

Acknowledgment

The research leading to these results is funded by the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA) and by the Ministry of Science and Technology of Spain under contract TIN2015-65316-P. Miloš Panić is funded by the Spanish Ministry of Education under the FPU grant FPU12/05966. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness and FEDER funds through grant TIN2014-60404-JIN. Jaume Abella is partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

References

[1] *ARM AMBA 3 AXI Specification (available at http://www.arm.com/products/solutions/axi_spec.html)*.
[2] *NanoC: NaNoC design platform. http://www.nanoc-project.eu*.
[3] Soclib, http://www.soclib.fr/trac/dev, 2012.
[4] ARINC Inc. *ARINC Report 651-1: Design Guidance for Integrated Modular Avionics*, 1997.
[5] AUTOSAR consortium. AUTomotive Open System ARchitecture (AUTOSAR). Standard v4.1, 2014.
[6] E. Bolotin, et al. Qnoc: Qos architecture and design process for network on chip. *JOURNAL OF SYSTEMS ARCHITECTURE*, 2004.
[7] A. Burns, et al. A wormhole noc protocol for mixed criticality systems. In *RTSS*, 2014.
[8] D. Dasari, et al. Noc contention analysis using a branch-and-prune algorithm. *ACM Trans. Embed. Comput. Syst.*, 13(3s), March 2014.
[9] J. Duato, et al. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2002.
[10] José Duato, et al. Mmr: A high-performance multimedia router - architecture and design trade-offs. In *HPCA*, 1999.
[11] T. Ferrandiz, et al. A sensitivity analysis of two worst-case delay computation methods for spacewire networks. In *ECRTS*, 2012.
[12] Jos Flich and Davide Bertozzi, editors. *Designing network on-chip architectures in the nanoscale era*. Chapman & Hall/CRC computational science series. Chapman and Hall/CRC, 2011.
[13] M. D. Gomony, et al. A generic, scalable and globally arbitrated memory tree for shared DRAM access in real-time systems. In *DATE*, 2015.

[14] K. Goossens, et al. Aethereal network on chip: concepts, architectures, and implementations. *Design Test of Computers, IEEE*, 2005.

[15] J. Jalle, et al. Deconstructing bus access control policies for real-time multicores. In *SIES 2013*, 2013.

[16] H. Kashif, et al. ORTAP: an offset-based response time analysis for a pipelined communication resource model. In *RTAS*, 2013.

[17] B. Kim et al. A real-time communication method for wormhole switching networks. In *ICPP*, 1998.

[18] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, 2001.

[19] Sunggu Lee. Real-time wormhole channels. *Journal Of Parallel And Distributed Computing*, 63:299–311, 2003.

[20] P. Munk, et al. Dynamic guaranteed service communication on best-effort networks-on-chip. In *PDP*, 2015.

[21] B. Nikolic, et al. Worst-case communication delay analysis for many-cores using a limited migrative model. In *RTCSA*, 2014.

[22] R. Obermaisser, et al. The time-triggered system-on-a-chip architecture. In *ISIE*, 2008.

[23] M. Panic, et al. Parallel many-core avionics systems. *EMSOFT*, 2014.

[24] M. Paolieri, et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.

[25] Mathieu Patte and Vincent Lefftz. System impact of distributed multi core systems. Technical Report ESTEC Contract 4200023100, ESA, 2011.

[26] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[27] P. Puschner, et al. Towards composable timing for real-time software. In *1st International Workshop on Software Technologies for Future Dependable Distributed Systems. 2009*.

[28] Y. Qian, Z. Lu, and W. Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *IEEE/ACM NoCS*, 2009.

[29] D. Rahmati, et al. Computing accurate performance bounds for best effort networks-on-chip. *IEEE Transactions on Computers*, 2013.

[30] J. Rattner. *Single-chip Cloud Computer: An experimental many-core processor from Intel Labs*.

[31] A. Roca, et al. VCTlite: Towards an efficient implementation of virtual cut-through switching in on-chip networks. In *HiPC 2010*, 2010.

[32] M. Schoeberl, et al. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *IEEE/ACM NoCS*, 2012.

[33] Zheng Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *NoCS*, 2008.

[34] Zheng Shi and Alan Burns. Real-time communication analysis with a priority share policy in on-chip networks. *ECRTS*, 2009.

[35] J. Sparsoe. Design of networks-on-chip for real-time multi-processor systems-on-chip. In *ACSD*, 2012.

[36] Tilera. *TILE-Gx Processors Family http://www.tilera.com/products/TILE-Gx.php*.

[37] A. Wilson and T. Preyssler. Incremental certification and Integrated Modular Avionics. In *DACS*, 2008.