Computing Accurate Performance Bounds for Best Effort Networks-on-Chip

Dara Rahmati, *Student Member*, *IEEE*, Srinivasan Murali, *Member*, *IEEE*, Luca Benini, *Fellow*, *IEEE*, Federico Angiolini, *Member*, *IEEE*, Giovanni De Micheli, *Fellow*, *IEEE*, and Hamid Sarbazi-Azad, *Member*, *IEEE*

Abstract—Real-time (RT) communication support is a critical requirement for many complex embedded applications which are currently targeted to Network-on-chip (NoC) platforms. In this paper, we present novel methods to efficiently calculate worst case bandwidth and latency bounds for RT traffic streams on wormhole-switched NoCs with arbitrary topology. The proposed methods apply to best-effort NoC architectures, with no extra hardware dedicated to RT traffic support. By applying our methods to several realistic NoC designs, we show substantial improvements (more than 30 percent in bandwidth and 50 percent in latency, on average) in bound tightness with respect to existing approaches.

Index Terms—SoC, NoC, performance, QoS, best-effort analysis, real time, analytical model, wormhole switching

1 Introduction

The Network-on-Chip [1], [2] paradigm has emerged in recent years to overcome the power and performance scalability limitations of point-to-point signal wires, shared buses, and segmented buses [1], [2], [3], [4], [5], [6]. While the scalability and efficiency advantages of NoCs have been demonstrated in many occasions, their timing predictability and suitability to transport real-time communication are still a source of technical concern.

Many applications have strict requirements on latency and bandwidth of on-chip communication, which are often expressed as real-time constraints on traffic flows. On a NoC fabric, this translates to guaranteed quality of service (QoS) requirements for packet delivery. Different approaches have been used to support guaranteed QoS for NoCs: priority-based switching schemes [7], time-triggered communication [8], time-division multiple access [9], and many variations thereof. All these approaches require the use of special hardware mechanisms and often come with strict service disciplines that limit NoC flexibility and penalize the average performance to provide worst-case guarantees. In fact, NoC prototypes are often classified as being either *best- effort* (BE) or *guaranteed-service* (GS), depending on the availability of hardware support for RT traffic.

Our work takes a new viewpoint. We consider best-effort NoC architectures without special hardware support for QoS traffic. We only assume that the traffic injected by the network's end-nodes is characterized in terms of worst case behavior. We then formulate algorithms to *find latency and bandwidth bounds on end-to-end traffic flows transported by a best-effort wormhole NoC fabric with no special hardware support for RT traffic*. For applications with traffic streams that have RT latency and/or bandwidth constraints, it is critical to be able to bound the maximum delay and minimum injectable bandwidth for packets of such streams. This helps in choosing topologies that meet the RT constraints with minimum area, power overhead and optimum utilization of resources. Our approach is inspired by the work by Lee et al. [10] for traditional multiprocessor networks, and extends it in several directions. We propose two different methods for characterizing worst case performance. The first method, Real-Time Bound for High-Bandwidth traffic (RTB-HB), is conceived for NoCs supporting workloads where injected flows have high demands of average bandwidth and require a guaranteed worst traffic minimum bandwidth (m*BW*) and maximum upper bound NoC traversal latency (UB). In this case, we do not assume any a priori regulation on the traffic injection rate; a core can send packets at any time, as long as the network has buffer capacity to accept them.

The second method considers applications with latency-critical flows that require low and guaranteed UB values, but have moderate bandwidth requirements, and thus can send packets at intervals no shorter than a minimum permitted interval—which obviously implies a maximum bandwidth (*MBW*) limitation. This method, called Real-Time Bound for Low-Latency traffic (RTB-LL) requires a very simple traffic regulation at network injection points. RTB-LL is a significant improvement to the WCFC bound

- *D. Rahmati is with the HPCAN, the Department of Computer Engineering, Sharif University of Technology, Room 701, Azadi Avenue, Tehran, Iran. E-mail: d_rahmati@ce.sharif.edu.*
- *S. Murali and F. Angiolini are with the iNoCs Sarl, Lausanne and the EPFL, INF 331, Station 14, Lausanne-CH 1015, Switzerland. E-mail: {murali, angiolini}@inocs.com.*
- *L. Benini is with the Micrel Lab @ DEIS - Dipartimento di Elettronica, Informatica e Sistemistica, Facoltà di Ingegneria, Università di Bologna, Viale Risorgimento 2, Bologna 40136, Italy. E-mail: luca.benini@unibo.it.*
- *G. De Micheli is with the Institute of Electrical Engineering and the Integrated Systems Centre, Ecole Polytechnique Federale de Lausanne (EPFL), INF 341, Station 14, Lausanne-CH 1015, Switzerland. E-mail: giovanni.demicheli@epfl.ch.*
- *H. Sarbazi-Azad is with the Department of Computer Engineering, Sharif University of Technology, Room 621, Azadi Avenue, Tehran, Iran. E-mail: azad@sharif.edu.*

Manuscript received 24 Jan. 2011; revised 15 Nov. 2011; accepted 29 Nov. 2011; published online 14 Dec. 2011.
Recommended for acceptance by Y. Yang.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-01-0051.
Digital Object Identifier no. 10.1109/TC.2011.240.

TABLE 1
Typical Upper Bound Delay and Bandwidth in Different Methods

Methods	RTB-LL (LL), RTB-HB (HB), WCFC (W)
Upper Bound Delay Comparison	$UB_{LL} \leq UB_{HB} \leq UB_W$
Maximum Permitted and minimum Guaranteed Bandwidth Comparison	$MBW_{LL} \geq mBW_{HB} \geq MBW_W$

proposed in [10], while RTB-HB is completely new. Table 1 compares typical values for upper bound delay and bandwidth of RTB-HB, RTB-LL, and WCFC methods. In [11] we presented these methods in their basic modes of operation. In this paper, we extend the methods to be more comprehensive, considering more generic NoC models, supporting various modes of operation and more experiments. In particular, we have several new and important contributions from our earlier work in [11].

The remainder of the paper is organized as follows: Section 2 summarizes related work. Section 3 gives definitions and basic concepts. Section 4 describes RTB-HB and RTB-LL methods. Section 5 focuses on experimental results and quantitative comparisons. Section 6 describes the time complexity of the proposed methods. Finally, Section 7 concludes the paper.

2 Related Work

The body of knowledge on macroscale RT networks is extensive and an overview of the state of the art is beyond the scope of this work. The interested reader is referred to [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. Here, we focus on RT-NoCs, which have often been called guaranteed-service or QoS-enabled NoCs.

QoS is an important issue in many application domains such as multimedia, aerospace, healthcare, and military. Many of these applications have one or more traffic flows that have real-time requirements and need hard QoS guarantees. Two major parameters that account for QoS guarantees in NoC are *worst case delay* and *worst case bandwidth*. They are sometimes referred as *upper bound delay* and *lower bound bandwidth of flows*. Historically, designers have focused on extracting the average delay and average bandwidth, and a large body of work to extract such parameters exists [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33]. Simulation and mathematical modeling are two different approaches to do so. While simulation is widely used in many situations, it is time consuming and it gives limited insight on sensitivity to traffic parameters and worst case conditions. In contrast, devising accurate mathematical models of a system is complicated, but if such models can be extracted, they are usually computationally efficient and insightful. Therefore, they can be used within design tools, for example, to iterate in the NoC synthesis process to tailor NoC architectures for specific applications. Frequently used mathematical frameworks are queuing theory and statistical timing analysis [25], [26], [28], [31]. A node is modeled as a queuing system, which can be $M/G/1, M/G/1/t, G/G/1$, etc. The NoC then is modeled by interconnecting a number of queues and the parameters are then extracted using standard solutions from queuing theory. In these models, the applicability and accuracy are the main concerns.

In order to provide QoS, some NoC architectures use special hardware mechanisms. They are known as Guaranteed Service NoCs, as opposed to Best Effort NoCs. To distinguish briefly, GS NoCs commit to a performance level for one or more flows (typically latency or bandwidth). *Hard* or *soft* QoS can be provided, depending on whether the NoC actually strictly enforces the desired performance level or merely strives to achieve it. GS NoCs can leverage *resource reservation* or *priority-based scheduling mechanisms*. The former technique usually achieves hard QoS, but the resource utilization may be poor because reserved resources are underutilized. The latter may achieve better resource utilization as resources are used on demand, like a best effort fashion with priority, but generally only ensures soft QoS and problems like starvation of low priority flows may occur. GS NoCs require extra hardware complexity with respect to BE NoCs to support redundant resources or priority mechanisms. On the other hand, the performance of flows in a GS NoC can be more easily characterized. In a pure BE NoC, analyzing the temporal behavior of the flows is very complex, due to the large number of contentions that may block a packet of a flow several times along its journey to the destination. Also simulation is very complicated and time consuming, as identifying the worst case scenario, and enforcing the network to operate in such worst case situation, is extremely difficult, if not impossible. Thus, the modeling approach may be an option; however, due to extreme complexity, until recently there have been no applicable approaches to model the performance parameters in worst case situations. Thus, most of the efforts to provide QoS have been in the context of GS or combinations of GS and BE NoCs.

In [9], Goossens et al. present the Æthereal NoC which combines GS with BE. The MARS [34], aSoc [35], and Nostrum [36] architectures use time division multiplexing (TDMA) mechanisms to provide real-time guarantees on packet-switched networks. The aelite NoC [37] provides a GS and scalable TMDA-based architecture, using mesochronous or asynchronous links. In Shi and Burns [7], a priority-based wormhole switching for scheduling RT flows is presented. In [8], Paukovits and Kopetz propose the concept of a predictable Time-Triggered NoC (TTNoC) that realizes QoS-based communication services. Diemer and Ernst in [38] introduce Back Suction, a flow control scheme to implement service guarantees using a prioritized approach between BE and GS services. In [39], Hansson et al. provide the latency and throughput guarantees based on the approach of data flow analysis technique, determining required buffer size at network interfaces, which is applicable to the Æthereal NoC. Many other works have been published with variations over these basic ideas [40], [41], [42], [43], [44], [45], [46], [47], [48], [49].

However, most NoC architectures are of *best effort* type [50] and do not have special hardware mechanisms to guarantee QoS. Today, to the best of our knowledge, there are only few works that calculate worst case bandwidth and delay values for a BE NoC. In [51], the lumped link model was proposed where the links a packet traverses are lumped into a single link. This model does not distinguish direct contention (due to arbitration losses) from indirect contention (due to full buffers ahead along the path), thus the estimated bounds are pessimistic. In [52], Qian et al. provide a method based on network calculus [53], [54] to

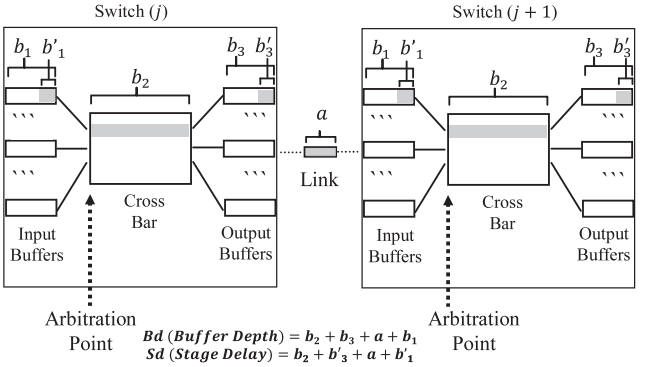


Fig. 1. Switch model and parameters.

calculate real-time bounds for NoCs; the method uses service curves and arrival curves that characterize the service characteristics of switches and injected traffic. Extracting arrival curves is not a straightforward task for many applications. Thus, for an arbitrary injected traffic load, traffic regulators may be needed to make sure that the amount of injected traffic in a specified period does not exceed a specified level. In [55] a buffer optimization problem is solved under worst case performance constraints based on network calculus. In [56], Bakhouya et al. also present a model based on network calculus to estimate the maximum end-to-end and buffer size for mesh networks; the delay bounds calculated for the flows are not hard bounds and real values may be larger.

In [11], we proposed some methods for worst case analysis that do not need traffic regulators. The bounds presented by those methods are tighter than those reported by previous studies. In this paper, we have extended these methods in several directions: we provide a more detailed switch model to differentiate *stage delay* and *buffer depth*; this results in tighter bounds in calculations of RTB-LL method. Our analysis also considers networks with virtual channels and variable buffer lengths; the analysis for small buffers is also extended to account for message lengths, which results in tighter bounds with respect to our previous results.

3 The Network Model

A router model is essential to characterize network latency and bandwidth. We consider the very general reference architecture shown in Fig. 1 where a crossbar handles the connections among input and output channels inside the router. For more generality, we consider optional buffering at input and output ports. We assume round-robin arbitration in the switches, a commonly used arbitration scheme in many NoCs. Each port is equipped with some virtual channels sharing the bandwidth of the physical channel associated with it. Links, which can be pipelined to maximize the operating frequency, connect the output ports to the input ports of adjacent routers. Note that due to backpressure signaling, packet loss and packet dropping do not happen in switches. Table 2 summarizes the parameters used to describe the model. For the sake of simplicity, we use a single parameter *Freq* for the operating frequency of all cores and *FlitWidth* as the data width of all NoC links.

The *buffer depth* (B_d) parameter is used in the paper frequently. As seen in Fig. 1, B_d is the summation of a number of *registers* and/or of the number of slots of one or two *FIFO*(s),[1] from the arbitration point (at the entry of the crossbar) of switch j to the arbitration point of switch $j+1$. The input buffer depth is denoted by b_1 (we assume at least one register), b'_1 is the minimum delay of the header flit of a packet in the input buffer of a switch, b_2 is the number of cycles to traverse a switch crossbar (if pipelined), b_3 is the depth of the output buffer (if any), b'_3 is the minimum delay of the header flit of a packet in the output buffer of a switch and a is the number of registers (if any) along a link to compensate the propagation delay of the wires. Note that b_2 and a represent latencies that packets face even in the absence of congestion, while b_1 and b_3 become most relevant in case of blocking, when buffers fill up; in the absence of congestion, input and output buffers can be traversed in a single cycle instead (b'_1 and b'_3). Blocking always happens because of arbitration conflicts, either directly in front of a switch crossbar, or indirectly due to full buffers ahead. For simplicity, throughout the paper, we consider the buffering between two adjacent switches to be lumped, so we mention "*output buffer of switch j*" and "*input buffer of switch* $j+1$" equivalently, referring to the same number of intermediate registers or FIFOs between the arbitration points of switches j and $j+1$, i.e., to B_d. The stage delay (S_d) parameter instead describes the minimum delay the header flit of a packet will face in the absence of any contention with other packets, between arbitration points of two adjacent switches. Please also note that the switches along a path are indexed $j = 1..m$, but $j = 0$ can

TABLE 2
Network Parameters, Symbols, and Used Notation

Parameter	Description
$Freq$	Clock frequency of the system
a	Number of pipeline stages (registers) used to segment NoC links to compensate the propagation delay of the wires
b_1	Depth of switch input buffer, as a FIFO
b'_1	Delay of packet header in switch input buffer without contention (1) If output buffer is presented: $b'_1 = 1 (b'_1 \leq b_1)$ (2) If output buffer is not presented: $b'_1 = b_1 = 0$
b_2	Number of pipeline stages of the crossbar (0 if combinational)
b_3	Depth of switch output buffer, as a FIFO (0 if none)
b'_3	Delay of switch output buffer without contention, (1) If output buffer is presented: $b'_3 = 1 \ (b'_3 \leq b_3)$ (2) If output buffer is not presented: $b'_3 = b_3 = 0$
b	$\triangleq b_1 + b_2 + b_3$
b'	$\triangleq b'_1 + b_2 + b'_3$
B_d	Buffer depth parameter $\triangleq a + b_1 + b_2 + b_3 = a + b$ The summation of the number of registers or FIFO(s) from the arbitration point (at the entry of the crossbar) of switch j to the arbitration point of switch $j+1$
S_d	Stage delay $\triangleq a + b'_1 + b_2 + b'_3 = a + b'$ The delay of header flit of a packet that has arrived at the arbitration point of switch j to arrive at the arbitration point of switch $j+1$ in the absence of contention. A stage comprises a switch and a link.
ts_1	Latency overhead for injecting a packet into the network
ts_2	Latency overhead for ejecting a packet at the destination
$FlitWidth$	Width of NoC links, in bytes
SW_j	j-th switch of the network

1. Throughout this paper we assume *registers* to be clocked pipeline registers, in which to traverse a number of n cascaded registers, n clock cycles are always required. A *FIFO* is the traditional First-In, First-Out queue. In case a flit traverses an empty FIFO of length n, in the absence of traffic contention, it will face one cycle and in case of a filled FIFO, will face n cycles of delay. We use the general term *buffer* throughout the paper to address both types of registers and FIFOs or a combination of them.

TABLE 3
Traffic Model Parameters

Parameter	Description
F_i	i-th traffic flow in the network
L	General terminology for length of packets, in flits
L_i	Length of the packets of F_i, in flits
S_i	Source node of F_i
D_i	Destination node of F_i
P_i	Packet of F_i
h_i	Number of switches (hops) along the path of F_i

TABLE 4
Performance Model Parameters

Parameter	Description
UB_i	Upper bound delay for a packet P_i of F_i to traverse the NoC
MI_i	Maximum interval until the ability to inject a new packet of F_i, used in method RTB-HB
mI_i	Minimum permitted interval between two consecutive packets of F_i, used in methods WCFC and RTB-LL
mBW_i	Worst-traffic minimum injectable bandwidth for F_i, used in method RTB-HB and is equal to: $mBW_i \triangleq L \times FlitWidth / MI_i \times Freq$
MBW_i	Maximum permitted bandwidth for F_i, used in methods WCFC and RTB-LL and is equal to: $MBW_i \triangleq L \times FlitWidth / mI_i \times Freq$
u_i^j	The time needed for P_i to go from the input buffer of $SW_j (j > 0)$ or from generating process in $S_i (j = 0)$ to the input buffer of $SW_{j+1} (0 \leq j \leq hi - 1)$ or $D_j (j = hi)$
U_i^j	The time needed for P_i to go from the output buffer of $SW_j (j > 0)$ or from the output buffer of $S_i (j = 0)$ to the output buffer of SW_{j+1}. This parameter is used for convenience and, mathematically it can be written based on u_i^j, as $U_i^j = u_i^{j+1}, j \geq 0$
$z_0(i,0)$	Number of flows contending with F_i at S_i
$z_c(i,j)$	Number of flows contending with F_i at SW_j at output channel c
$I(x)$	Index of the x-th flow contending with F_i at S_i(or SW_j)

conveniently be seen as a *virtual switch* inside the source node, which acts like a physical switch to model source conflicts (i.e., sending more than one flow from a source node). The parameters ts_1 and ts_2 model the setup time at NoC sources and consumption time at NoC destinations to inject and eject packets. Of course, to be able to use finite parameters, we assume that the receiving nodes are able to accept incoming data at the required rates.

4 Network Traversal Delay Analysis

Table 3 lists the parameters we use to describe traffic flows across the network, while Table 4 summarizes the parameters that we use to model the performance of such flows. Most notably, UB_i represents the upper bound delay for a packet of flow F_i traversing the network, and is a key factor for the interconnect designer. We try to use a notation as close as possible to that used in [10] for ease of comparison. We first present a method called Real Time Bound for High-Bandwidth traffic which calculates UB_i in a completely worst case traffic situation. Crucially, this *includes the possibility for other system cores to inject unregulated bandwidth, i.e., any amount of traffic at irregular intervals*. This is a key property for real-world interconnects analysis, as most available IP cores operate on an unregulated-injection basis. In order to calculate UB_i in such a case, we consider all intermediate buffers along the route to be full, and we assume arbitration loss at all switches where other flows are contending for the same output port. As it will be seen, the calculations always provide solutions for worst case situation on different flows that are unique due to the employed deterministic calculation procedure. The calculated values are tight in most network scenarios as the worst case situation can really happen. The bounds may be slightly pessimistic in rare network scenarios where some of the contending flows connect two switches on different routes that can prevent providing enough contending packets to create worst situations.

Deadlock and livelock do not occur as we assume the routing path along the switches for all the flows are deterministic and predefined (like the networks used in [57]). As we are modeling the worst case behavior, we consider that the flows send the packets at maximum possible rate that the network permits; thus at this level of design, knowing the flow behavior is not important. Since switches are assumed to feature round-robin arbitration, even though we assume the current flow to be serviced last, the maximum delay is bounded, i.e., starvation cannot occur. Therefore, the packets sent by the source S_i are eventually delivered. RTB-HB calculates the Maximum Interval MI_i, i.e., the number of cycles after which the output buffer of S_i is guaranteed to be free again for further injection. From this value, the worst traffic minimum injectable bandwidth (mBW_i) can also be easily derived. This analysis can be applied to most NoC architectures, without any specific QoS hardware or software provisioning. We then move on to the description of another method, called RTB-LL. In this scenario, we assume that traffic injection can be regulated, as in some application scenarios. Therefore, we also calculate a minimum permitted interval (mL_i) between two consecutive packets from the same source, which can be translated into a maximum permitted bandwidth (MBW_i). This approach is similar to the previously reported method [10] (*Real-time wormhole channel feasibility checking* or WCFC, which will be briefly described later) but provides much better results in terms of bound tightness. For a proper operation, the system must then respect MBW_i bounds at runtime.

4.1 The Proposed Delay Model RTB-HB

The goal here is to calculate the parameters UB_i (worst case latency to traverse the network) and MI_i (maximum worst case interval). Let us first consider the case $B_d = L$.

4.1.1 The Case $B_d = L$

Note that $B_d = L$ means that a packet fills exactly the buffering resources between the arbitration points of two adjacent switches. Considering the case where the network is completely loaded (an unrealistic scenario just for visualization purposes) and $B_d = L$, the network operates by shuffling packets around in lockstep: all switches simultaneously rearbitrate every L cycles and packets trail each other, filling up the buffers as soon as they become free.

More formally, when P_i is generated in S_i, we consider all intermediate buffers along its route being full of packets from different flows. In the worst case, for P_i to reach its destination, all these packets must leave their buffers. Focusing on hop j, P_i may have arbitration conflicts with a

number $z_c(i,j)$ of other flows contending for the output channel c, e.g., flows F_a and F_r. Since round-robin arbitration is assumed, it is enough to consider all contending flows to send a packet before P_i to guarantee a worst case analysis. The order in which contending flows obtain the arbitration is not important for the latency calculation of P_i. So, P_a should make a one-hop forward progress. While P_a frees the buffers at hop j, flit by flit, the flits from P_r will smoothly replace the free buffer spaces. Eventually, P_i also goes through. Section 4.1.3 presents a simple example to visualize this. The parameter u_i^0 represents the time needed for P_i to be ejected from S_i and be placed in the output buffer of S_i (or input buffer of the first switch of F_i). u_i^j then represents the time needed for P_i to go from the input buffer of SW_j to the input buffer of SW_{j+1}, except for the last switch. At the last switch, P_i is ejected, so it is instead the time needed to get into the input buffer of destination D_i. To calculate UB_i, as shown in (1), all these contributions must be aggregated, plus the fixed overhead for the packet creation and ejection:

$$UB_i = ts_1 + ts_2 + \sum_j u_i^j, j = 0..h_i. \tag{1}$$

The time needed for S_i to inject the next packet is the time to create such a packet, plus the time needed for this packet to move on to the input buffer of the first switch. Thus,

$$MI_i = ts_1 + u_i^0. \tag{2}$$

To be consistent with the notations from [10], we introduce the uppercase U_i^j symbol, which models the hop delay from output buffer to output buffer (instead of from input buffer to input buffer).

Let us consider a packet of flow F_i initiated at the source S_i. For this packet to reach the input buffer of the first switch, all existing packets at that buffer have to leave. Such an existing packet could be a packet from the same flow F_i or any of the contending flows at the output channel of the source. Thus, the worst case time taken for any existing packet to leave the buffer is given by $MAX_x(U_i^0, U_{I(x)}^0)$, where $I(x)$ is the index of contending flows at the output channel, with $x = 1 \ldots z_0(i,0)$. Also, all other contending flows of F_i may have to send a packet before this flow. Thus, the total delay for a packet from F_i to reach the input buffer of the first switch is given by:

$$\begin{aligned} u_i^0 &= MAX_x\left(U_i^0, U_{I(x)}^0\right) + \sum_x U_{I(x)}^0, \\ x &= 1..z_0(i,0). \end{aligned} \tag{3}$$

Similarly, for the subsequent hops, u_i^j can be calculated as:

$$\begin{aligned} u_i^j &= MAX_x\left(U_i^j, U_{I(x)}^j\right) + \sum_x U_{I(x)}^j, \\ x &= 1..z_c(i,j),\ 1 \le h_i \le h_i. \end{aligned} \tag{4}$$

Please note that, if there is no contention for the flow, the above equation reduces to $u_i^j = U_i^j$ This is again akin to a packet moving in a pipeline fashion in the network.

In order to calculate U_i^j values, let us consider the packet from flow F_i moving from output buffer of the source to the output buffer of the first switch. For the packet to move, any existing packet from the output buffer of the first switch should move to the output buffer of the second switch. Similar to the above calculations, the maximum delay is given to be $MAX_x(U_i^{j+1}, U_{I(x)}^{j+1})$. Please note the small difference from the u_i^j calculations that, in this case, the values of U_i^j at a switch (j) depends on the values at the next switch $(j+1)$ on the path. The U_i^j values can be obtained as

$$\begin{aligned} U_i^j &= MAXx\left(U_i^{j+1}, U_{I(x)}^{j+1}\right) + \sum_x U_{I(x)}^{j+1}, \\ x &= 1..z_c(i,j+1), \quad 0 \le j \le h_i - 1. \end{aligned} \tag{5}$$

For the case of the last switch, from the output port, the packet can be ejected in L_i cycles (one flit per cycle). Thus,

$$U_i^{h_i} = L_i, U_{I(x)}^{h_I(x)} = L_{I(x)}.$$

Based on (3) and (4), now the problem of finding UB_i and MI_i ((1) and (2)) is mapped onto a summation of U_i^j values, which can be solved by (5). Please note that we assume that the destination has enough buffers to eject the packets at the rate at which the network delivers them. By applying the above formulas recursively, we can obtain the worst case delay (UB) and injection rate (MI) for the different flows.

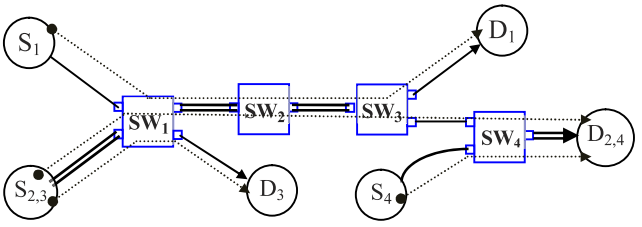


Fig. 2. A simple example network.

To describe the details of different aspects of analytical method RTB-HB for the calculation of upper bound delay and interval, we apply them step-by-step to an example NoC (shown in Fig. 2). The NoC contains four switches and there are four message flows from S_1 to D_1, $S_{2,3}$ to D_3, $S_{2,3}$ to $D_{2,4}$, and S_4 to $D_{2,4}$ ($S_{2,3}$ and $D_{2,4}$ are source and destination with two flows originated from or finished at them). We consider $B_d = L = 4$, in this example.

As an example, we study the time needed for a packet P^0 of flow F_1 to cross the network. In general, from (1), we can write: $ts_2 + \sum_j u_1^j, \ \ j = 0..3$.

To start, let us model the time u_1^0 needed to move from S_1 to the input buffer of switch SW_1. We start from the most congested network possible, so there exists another packet P^1 of the same flow ahead, and this packet needs U_1^0 to move from the output buffer of the source (remember that source nodes are tagged with superscript 0) to the output buffer of SW_1 ; so, $u_1^0 = U_1^0$. U_1^0 has to be calculated recursively based on the delays of the contending packets and delays of the packets ahead along the same route.

We observe that two factors mainly contribute when calculating the delay: first, the possibility of losing arbitrations at SW_1 ; second, the fact that there may be no available buffer space at the output of SW_1 (due to arbitration losses ahead), which also effectively stalls packets at the input of SW_1. For what concerns the arbitration loss, it can be seen

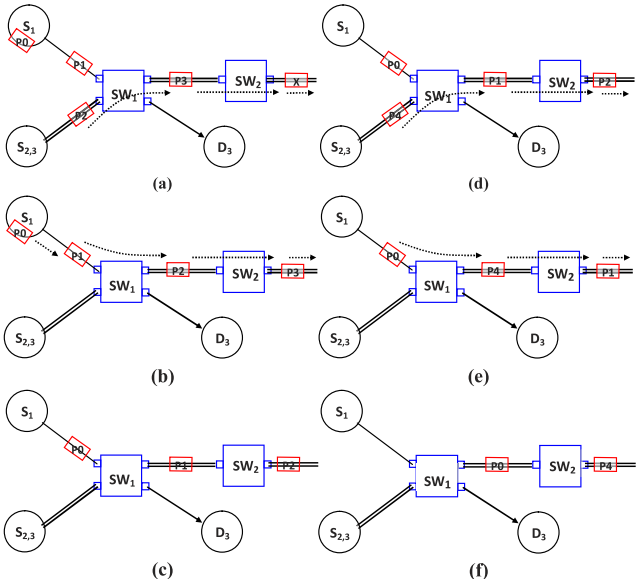


Fig. 3. (a, b, c) packet P^0 goes from a process that generates it (F_1 in S_1) to the input buffer of SW_1 (d, e, f) packet P^0 goes from input buffer of SW_1 to input buffer of SW_2.

that flow F_1 contends with flow F_2 at the output of SW_1. Thus, a packet P^2 of F_2 currently in the input buffer of SW_1 could be arbitrated before P^1. For what concerns the output buffer full condition, in the worst case, there will be a single ($B_d = L$) packet P^3 in the output buffer of SW_1. P^3 could belong to either F_1 or F_2, where, respectively, U_1^1 or U_2^1 models the time for such a packet to move ahead, from the output buffer of SW_1 to the output buffer of SW_2. $MAX(U_2^1, U_1^1)$ models the worst case delay affecting the flow under study. During the time $MAX(U_2^1, U_1^1)$, the packet P^1 moves on to the output buffer of SW_2, leaving the output buffer at SW_1 empty. However, in the worst case, an arbitration loss occurs to P^1, so it is packet P^2 which will smoothly replace P^3 (Fig. 3a). Before P^1 can move on by one hop, we must also consider the time for packet P^2 to go from the output buffer of SW_1 to the output buffer of SW_2 (Fig. 3b), which is U_2^1. Thus, we can write:

$$u_1^0 = U_1^0 = MAX\left(U_2^1, U_1^1\right) + U_2^1,$$

which traces back to (3). As mentioned above, this is the delay for P^1 to move one hop on, but equivalently is also the delay for P^0 to replace it in the previous location (Fig. 3c).

Now, similarly, P^0 needs to move another hop on, from the input buffer of SW_1 to the input buffer of SW_2, with a delay which is defined as u_1^1. It is possible to use the relation $u_1^1 = U_1^0$ based on the equations described in previous section, but for clarity, we always describe u_i^j based on U_i^j. As shown in Figs. 3d, 3e, and 3f), in the worst case, packet P^0 should wait for a packet P^4 of F_2. A packet P^1, again either from F_1 or F_2, should be considered at the output buffer of SW_1. So again, during the time $MAX(U_2^1, U_1^1)$, while P^1 moves on to the output buffer of SW_2, P^4 will replace it. P^4 itself then takes U_2^1 to move on, allowing P^0 to eventually get to the input buffer of SW_2. Thus,

$$u_1^1 = MAX\left(U_2^1,\ U_1^1\right) + U_2^1.$$

$u_1^0 = U_1^0$ $u_1^1 = MAX(U_2^1, U_1^1) + U_2^1$ $u_1^2 = MAX(U_2^2, U_1^2)$ $u_1^3 = U_1^3$	$u_2^0 = MAX(U_2^0, U_3^0) + U_3^0$ $u_2^1 = MAX(U_2^1, U_1^1) + U_1^1$ $u_2^2 = MAX(U_2^2, U_1^2)$ $u_2^3 = U_2^3$ $u_2^4 = U_4^4 + U_2^4$
$U_1^3 = U_1^3 = L_1$ $U_2^2 = U_2^3 = U_4^4 + U_2^4 = L_4 + L_2$ $U_1^1 = MAX(U_1^2, U_2^2) = L_4 + L_2$ $U_2^1 = MAX(U_1^2, U_2^2) = L_4 + L_2$ $U_1^0 = MAX(U_2^1, U_1^1) + U_2^1 = 2L_2 + 2L_4$	$U_2^0 = MAX(U_2^1, U_1^1) + U_1^1$ $= 2L_2 + 2L_4$ $U_3^0 = U_3^1 = L_3$ $UB_2 = 7L_2 + L_3 + 7L_4$ $MI_2 = 2L_2 + L_3 + 2L_4$
$UB_1 = L_1 + 5L_2 + 5L_4$, MI_1 $= 2L_2 + 2L_4$	
$u_3^0 = MAX(U_2^0, U_3^0) + U_2^0$ $u_3^1 = U_3^1 = L_3$	$u_4^0 = U_4^0$ $u_4^4 = U_2^4 + U_4^4$
$UB_3 = 4L_2 + L_3 + 4L_4$ $MI_3 = 4L_2 + 4L_4$	$U_4^0 = U_2^4 + U_4^4 = u_4^4$ $UB_4 = 2L_2 + 2L_4, MI_4 = L_2 + L_4$

Fig. 4. The complete calculation of UB_i and MI_i for the example NoC of Fig. 2 in method RTB-HB.

In a similar manner u_1^2 can be calculated. Once P^0 is in the input buffer of SW_3, it is only one hop away from its destination and as there is no contending flow at the destination, the ejection time for the messages equals L_1. For the sake of uniformity of presentation, we can write

$$u_1^3 = U_1^3 = L_1.$$

Now, the target metric UB_1 can be calculated recursively, as a function of U_i^j variables for the whole network starting from the last hop of each flow. The calculation of all intermediate values, e.g., U_1^1, U_2^1, U_1^2, U_2^2, U_1^3 and of the relevant metrics UB_i and MI_i, is shown in Fig. 4. Please note that to speed up the recursion steps, intermediate values can be calculated only once and then stored and used later.

When considering the source $S_{2,3}$ it can be noticed that two flows F_2 and F_3 can originate from it; therefore, source conflicts may happen. As Fig. 4 shows, for example, when analyzing flow F_2, u_2^0 (the time to transfer a packet of F_2 into the input buffer of the first switch) should include a delay $MAX(U_2^0, U_3^0)$, which accounts for a packet of either F_2 or F_3 to move away from the input of SW_1 toward the input of SW_2 (during which time we must assume, in the worst case, that it is a packet of F_3 which replaces it), and then again the time U_3^0 for this latter packet to also move on, and finally letting a packet from F_2 in. The calculation for u_3^0 is done similarly.

4.1.2 The Case $B_d < L$

The same values calculated for the case $B_d = L$ can be used for the case $B_d < L$. Assume network A with $B_d < L$ and a completely equivalent network B but with $B_d = L$. Let us consider $UB_i(A)$ and $UB_i(B)$ to be the upper bound delays for these networks. It is possible to use value of $UB_i(A)$ instead of $UB_i(B)$ for network A. The reason is simply that the number of contending flows along the path of a flow is the same in both the networks A and B, but the number of in-flight packets in the worst case situation is larger for network B.

In order to tighten the performance bounds, we also present equations to calculate $UB_i(A)$ directly, achieving lower figures than those calculated above for network B. For this purpose, we introduce a new parameter δ_i^j and extend the definitions of u_i^j and U_i^j to support the case $B_d < L$ for network A ; the new definitions comply with those used for the case $B_d = L$.

Definition 1. *We denote by δ_i^j the worst case delay (in cycles) elapsing from the moment when the header flit of P_i enters the arbitration point of switch $j+1$ to the moment when the tail flit leaves the arbitration point of switch j.*

In particular, when $B_d = L$ (as in network B), this time is zero, since as soon as the header flit enters the arbitration point of switch $j+1$, the tail flit has left the arbitration point of switch j.

In order to calculate the parameters for network A, it is needed to calculate the values of δ_i^j for different i and j values. Here, for convenience, we consider L being divisible by B_d; thus, when there is a sufficiently long cascade of switches, a whole packet can fit between the arbitration points of the first and last switches. For this purpose, the number of required switches is $L/B_d + 1$ (2 when $B_d = L$). To calculate δ_i^j, since the header flit is at the arbitration point of switch $j+1$, $L - B_d$ flits of the packet have not passed the arbitration point of switch j, and therefore the header flit needs to traverse S more switches ($S = L - B_d/B_d = L/B_d - 1$), to ensure that the tail flit passes the arbitration point of switch j. If the number of remaining switches in the path of flow i is smaller than S, the packet simply exits the network at the destination node flit by flit, until the tail flit has left the arbitration point of switch j ; thus,

$$\delta_i^j = \begin{cases} \sum_{k=1}^{s} u_i^{j+k}(A) & \text{when } j+s \le h_i \\ \sum_{k=1}^{h_i-j} u_i^{j+k}(A) + (j+s-h_i) \times B_d & \text{when } j+s > hi, \\ \quad j = 0..h_i. \end{cases} \tag{6}$$

Definition2. *We denote by $U_i^j(A)$ the worst case delay elapsing from the moment when the header flit of P_i enters the arbitration point of switch $j+1$ to the moment when the tail flit leaves this point. When $B_d = L$ (as in network B), this definition matches the previous definition for $U_i^j(B)$ in (5).*

$$\begin{aligned} U_i^j(A) = {} & MAX_x\big(U_i^{j+1}(A) - \delta_i^{j+1},\ U_{I(x)}^{j+1}(A) - \delta_{I(x)}^{j+1}\big) \\ & + \sum_x U_{I(x)}^{j+1}(A) + \delta_i^{j+1}, \\ & x = 1..\ z_c(i, j+1),\ 0 \le j \le h_i - 1, \end{aligned} \tag{7}$$

and

$$U_i^{h_i}(A) = L. \tag{8}$$

In case $B_d < L$, a packet occupies more buffering space than that available between arbitration points of two adjacent switches. Consider now a situation where the header flit of a packet p of flow i is at the arbitration point of switch $j+1$, while an interfering packet q of flow t is also traversing switch $j+1$, but with its header already at the arbitration point of switch $j+2+S(S = L/B_d - 1)$. Also, a number $z_c(i, j+1)$ of packets at different input ports of switch $j+1$ are contending with p for the same output port. Before the tail flit of packet p can leave the arbitration point at switch $j+1$, the following must happen:

1. Packet q should proceed for B_d steps (registers), so that its tail flit leaves the arbitration point of switch $j+2$. After this step the header flit of one of the $z_c(i, j+1)$ packets should be at the arbitration point of switch $j+2$. This time is calculated as $u_t^{j+2+s}(A)$; it is easy to see that $u_t^{j+2+S}(A) = U_t^{j+1}(A) - \delta_t^{j+1}$. As t can be selected from any of i or $z_c(i, j+1)$ flows, the term to calculate the required time for this step is:
$$MAX_x\big(U_i^{j+1}(A) - \delta_i^{j+1}, U_{I(x)}^{j+1}(A) - \delta_{I(x)}^{j+1}\big), \quad x = 1..z_c(i, j+1).$$
2. All the $z_c(i, j+1)$ packets, to account for the worst case, shall be arbitrated before p; so, after this step the header of p is at the arbitration point of switch $j+2$. The required time for this step is $\sum_x U_{I(x)}^{j+1}(A)$ with $x = 1..\ z_c(i, j+1)$.
3. Packet p, whose header is at the arbitration point of switch $j+2$, must proceed for a number of steps until its tail flit leaves the arbitration point of switch $j+1$. This time is calculated as δ_i^{j+1}.

The summation of the time for these steps results in (7). As can be seen in (8), in the last switch for a flow, L cycles are required for the packet to be ejected to the destination.

Definition 3. *We denote by $u_i^j(A)$ the worst case delay elapsing from the moment when the header flit of P_i enters the arbitration point of $SW_j (1 \le j \le h_i - 1)$ to the time it enters the arbitration point of SW_{j+1} (for $j = 0$ it is the worst case delay elapsing from when the header flit of the packet in the source node is ready to be injected in the network to the time it enters the arbitration point of the first switch, also for $j = h_i$, it is the worst case delay elapsing from the moment the header flit of P_i enters the arbitration point of the last switch h_i to when a number of B_d flits of the packet have ejected the network at D_i.*

$$\begin{aligned} u_i^j(A) = {} & MAX_x\big(U_i^j(A) - \delta_i^j,\ U_{I(x)}^j(A) - \delta_{I(x)}^j\big) + \sum_x U_{I(x)}^j(A), \\ & x = 1..\ z_c(i, j),\ 0 \le j \le h_i - 1, \end{aligned} \tag{9}$$

and

$$u_i^{h_i}(A) = B_d + \sum_x U_{I(x)}^{h_i}(A),\ x = 1..\ z_c(i,\ h_i). \tag{10}$$

It is obvious that calculation of $u_i^j(A)$ is like $U_i^{j-1}(A)$ except for step 3 which is not required and term δ_i^j is omitted; thus, we can write:

$$U_i^{j-1}(A) = u_i^j(A) + \delta_i^j,\ j \ge 1. \tag{11}$$

Using the above definitions the equations to calculate $UB_i(A)$ and $MI_i(A)$ can be summarized as:

$$\begin{aligned} UB_i(A) = {} & ts_1 + ts_2 + \sum_j u_i^j(A) + (L - B_d), \\ & j = 0..\ h_i. \end{aligned} \tag{12}$$

In other words, the time needed for a packet to move from source to destination includes the source and destination overheads (ts_1 and ts_2), the time needed for the header flit to move across the network switch by switch, and $L - B_d$ (as for the last switch, (10) is used and the packet needs $L - B_d$ further steps to completely leave the network).

$u_1^3 = B_d$ $u_1^2 = MAX(U_2^2 - \delta_2^2, U_1^2 - \delta_1^2) = 2L - B_d$ $u_1^1 = MAX(U_2^1 - \delta_2^1, U_1^1 - \delta_1^1) + U_2^1$ $= 2L + B_d$ $u_1^0 = U_1^0 - \delta_1^0 = 2L - B_d$	$u_2^0 = MAX(U_2^0 - \delta_2^0, U_3^0 - \delta_3^0) + U_3^0 = 3L - B_d$ $u_2^1 = MAX(U_2^1 - \delta_2^1, U_1^1 - \delta_1^1) + U_1^1 = 2L + B_d$ $u_2^2 = MAX(U_2^2 - \delta_2^2, U_1^2 - \delta_1^2) = 2L - B_d$ $u_2^3 = U_2^3 - \delta_2^3 = B_d$ $u_2^4 = U_4^4 + B_d = L + B_d$
$L = 4, B_d = 2$ $S = L/B_d - 1 = 1$ $\delta_1^2 = u_1^3 = B_d$ $\delta_2^2 = u_2^3 = B_d$ $U_1^3 = U_1^3 = L$ $U_2^2 = U_2^3 - \delta_2^3 + \delta_2^3 = U_2^3 = 2L$ $U_2^3 = U_2^4 + U_2^4 = 2L$ $\delta_2^3 = u_2^4 = B_d + U_4^4 = B_d + L$ $U_1^2 = U_1^3 = L$ $U_1^1 = MAX(U_1^2 - \delta_1^2, U_2^2 - \delta_2^2) + \delta_1^2 = 2L$ $U_2^1 = MAX(U_2^2 - \delta_2^2, U_2^2 - \delta_2^2) + \delta_2^2 = 2L$ $\delta_2^1 = u_2^2 = 2L - B_d$ $U_1^0 = MAX(U_2^1 - \delta_2^1, U_1^1 - \delta_1^1) + U_2^1 + \delta_1^1$ $= 4L$ $\delta_1^1 = u_1^2 = 2L - B_d$ $\delta_1^0 = u_1^1 = 2L + B_d$	$U_3^0 = MAX(U_2^1 - \delta_2^1, U_1^1 - \delta_1^1) + U_1^1 + \delta_2^1 = 4L$ $U_3^0 = U_3^1 = L$ $\delta_2^0 = u_2^1 = 2L + B_d$ $\delta_3^0 = u_3^1 = B_d$ $UB_2 = 8L + B_d + (L - B_d) = 9L$ $MI_2 = 5L$
	$u_3^0 = MAX(U_2^0 - \delta_2^0, U_3^0 - \delta_3^0) + U_2^0 = 6L - B_d$ $u_3^1 = B_d$
	$UB_3 = 6L + (L - B_d) = 7L - B_d$ $MI_3 = 6L$
	$u_4^0 = U_4^0 - \delta_4^0 = L - B_d$ $u_4^4 = U_2^4 + B_d = L + B_d$
	$U_4^0 = U_2^4 + U_4^4 = 2L$ $\delta_4^0 = u_4^4 = U_2^4 + B_d = L + B_d$
$UB_1 = 6L + (L - B_d) = 7L - B_d$ $MI_1 = 4L$	$UB_4 = 2L + (L - B_d) = 3L - B_d,$ $MI_4 = 2L$

Fig. 5. The complete calculation of UB_i and MI_i for the example NoC of Fig. 2 with method RTB-HB when $B_d < L (L = 4\ ,\ B_d = 2)$.

To calculate the value of the maximum interval between two consecutive packets, it is enough to add the source overhead (ts_1), the time needed to send the header flit of the packet to the arbitration point of the first switch ($u_i^0(A)$), and the time needed for the last flit of the packet to leave the source node which is equal to δ_i^0. That is,

$$MI_i(A) = ts_1 + u_i^0(A) + \delta_i^0. \tag{13}$$

As seen in (9), in order to calculate $u_i^j(A)$, different values of δ_i^j should be calculated, which in turn requires $u_i^k(A)$ with $k > j$; so the order of calculation of $u_i^j(A)$ values should be from the larger switch indices toward the smaller ones, i.e., from destinations toward sources.

In order to show the implementation for the case $B_d < L$, Fig. 5 shows the case of $L = 4$ and $B_d = 2$ for the example scenario in Fig. 2. In order to calculate UB_1, the values u_1^3, u_1^2, u_1^1, and u_1^0 should be calculated. u_1^3 as described in (10) is equal to B_d as there is no flow contention on the port connected to D_1 in SW_3. Equation (9) is used to calculate u_1^2, which is equal to the time needed for the header flit of a packet P_1 of F_1 to move from the arbitration point of SW_2 to the arbitration point of SW_3. As there is no contention from flows from different input ports of SW_2 for the same output port, the term $\sum_x U_{I(x)}^j(A)$ in (9) is zero; thus, only the value of $MAX(U_2^2 - \delta_2^2,\ U_1^2 - \delta_1^2)$ is calculated (the reason for using the MAX operator is that the type of the packet just ahead of P_1 may be either from F_1 or F_2). It is obvious that flow contention happens between F_1 and F_2 for the output port of SW_1 which is connected to SW_2; thus, to calculate u_1^1 this contention should be considered and based on (9), the value is $MAX(U_2^1 - \delta_2^1,\ U_1^1 - \delta_1^1) + U_2^1$. To calculate u_1^0, as there is no source conflict, the term $\sum_x U_{I(x)}^j(A)$ is zero and the term $MAX(U_i^j(A) - \delta_i^j,\ U_{I(x)}^j(A) - \delta_{I(x)}^j)$ will result in $U_1^0 - \delta_1^0$. The calculation for other flows is done in the same manner.

4.1.3 The Case $B_d > L$

For simplicity, let us first consider the simple case $B_d = 2L$. In this case, exactly two packets fit between two adjacent switches along the path of flow F_i. It is possible to break the buffering space by imagining a dummy switch at the middle of input buffer of all the switches and rewrite the above formulas. For example, in Fig. 6, with the introduction

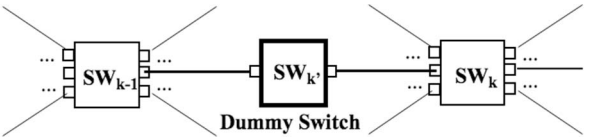


Fig. 6. Insertion of a dummy switch.

of dummy switches, the buffering space between switches k' and k becomes $B_d = L$.

In such a topology, we can write always $u_i^{k'} = u_i^k$; independent of the flows of the packets in the buffer thus, UB_i and MI_i are given as:

$$\begin{aligned} UB_i &= ts_1 + ts_2 + 2 \times \sum_j u_i^j,\ j = 0..h_i \\ MI_i &= ts_1 + u_i^0. \end{aligned} \tag{14}$$

Extending this approach to the case $B_d = m \times L,\ m - 1$ dummy switches can now be inserted between each pair of switches and thus:

$$\begin{aligned} UB_i &= ts_1 + ts_2 + m \times \sum_j u_i^j,\ j = 0..h_i \\ MI_i &= ts_1 + u_i^0. \end{aligned} \tag{15}$$

In cases where B_d is not a multiple of L, i.e.,

$$B_d = m \times L + k,\ 0 < k < m.$$

A straightforward conservative solution can round up B_d to $(m + 1) \times L$ and consider m dummy switches; so, the formula becomes

$$\begin{aligned} UB_i &= ts_1 + ts_2 + (m + 1) \times \sum_j u_i^j,\ j = 0..h_i \\ MI_i &= ts_1 + u_i^0. \end{aligned} \tag{16}$$

The above equation shows that MI_i, and thus mBW_i, is not a function of m. In other words, the minimum guaranteed bandwidth in RTB-HB does not depend on the network's buffering space when buffer depth is larger than message length. Indeed, in this scenario, we assume that a buffer of size $m \times L$ exists just before the arbitration point of the inputs of the switches and for the calculation purposes we assume $m - 1$ dummy switches are placed to split the buffer to m stages; as the delay for all the stages are equal, we can compare these stages in the network with a pipeline in which increasing or decreasing the number of stages does not affect the data injection rate into the pipeline although the delay will increase when m increases. This is true for the buffers just at the outputs of the source cores and thus the injection rate to the network is not affected when m changes. This is better shown in an example in the next section. Also as in Section 4.1.2 we have proposed the solution for parameterized packet Length of L and $B_d = 1$, it is possible to combine this solution with dummy switches solution to support arbitrary B_d on different switches (i.e., B_{d-j} on switch j). It is possible to consider spreading $B_{d-j} - 1$ dummy switches among the elements of a buffer of depth B_{d-j}, then reenumerate all the real and dummy switches in the

network and solve the equation for $B_d = 1$ (in the whole network) as suggested in Section 4.1.2.

The message length L_i can vary on a per-flow basis. To tackle this challenge, let us call the shortest message length $L_{\min}$. Let us first consider $B_d \leq L_{\min}$. In this case, it is intuitively possible to use the same equation as in Section 4.1.1. The only difference is that for every individual flow, L_i must be used as a parameter instead of L. Also, for $B_d \leq L_{\min}$ it is possible to use exactly the same approach used in Section 4.1.2. To further describe this, when $B_d = m \times L_{\min} + k$, with $0 < k < m$, it is possible to consider m dummy switches (or alternatively $B_d - 1$ dummy switches as described above) between each two consecutive switches and do the calculations as described in Section 4.1.2, using L_i values for different flows. Thus, based on the approach of variable message length described here, throughout the paper, we have used L_i instead of L.

4.1.4 Virtual Channels

The extension of the proposed method to support virtual channels is straightforward, provided that the index of the virtual channel used for a flow in different switches is a predefined parameter. For this purpose, all virtual channels in intermediate switches can be considered as physical channels in the proposed model. Arbitration conflicts would be among all the incoming virtual channels, and each output virtual channel would account for a separate output port (so, e.g., the value for $U_i^{h_i}$ would change from L_i to $m \times L_i$, assuming m virtual channels per physical channel). In the following equations, the modifications needed to support virtual channels are shown. The parameter $u_i^j[v_{i,j}]$ denotes the maximum delay to move from the arbitration point of switch j to the arbitration point of switch $j+1$ when using the predefined input virtual channel $v_{i,j}$ used for flow i at the input of switch j. Similarly, $U_i^j[v_{i,j+1}]$ is the maximum delay to move from the output buffer of switch j to the output buffer of switch $j+1$ using input virtual channel $v_{i,j+1}$ at the input of switch $j+1$. Also v_{i,h_i+1} is defined as the virtual channel index used for flow i at the input port of the destination (D_i) IP core. The parameter $u_i^0[1]$ is used to calculate MI_i as shown below; in this case, the selection of virtual channel number 1 is mandatory as only one flow enters the virtual switch inside an IP core

$$\begin{aligned}
u_i^j[v_{i,j}] &= MAX_x\big(U_i^j[v_{i,j+1}],\ U_{I(x)}^j[v_{I(x),j+1}]\big) \\
&\quad + \sum_x U_{I(x)}^j[v_{I(x),j+1}], \\
x &= 1..z_c(i,j),\ 0 \leq j \leq h_i \\
U_i^j[v_{i,j+1}] &= MAX_x\big(U_i^{j+1}[v_{i,j+2}],\ U_{I(x)}^{j+1}[v_{I(x),j+2}]\big) \\
&\quad + \sum_x U_{I(x)}^{j+1}[v_{I(x),j+2}], \\
x &= 1..\ z_c(i,j+1),\ 0 \leq j \leq h_i - 1 \\
U_i^{h_i}[v_{i,hi+1}] &= m \times L_i \\
UB_i &= ts_1 + ts_2 + \sum_j u_i^j[v_{i,j}],\ \ j = 0..h_i \\
MI_i &= ts_1 + u_i^0[1].
\end{aligned} \tag{17}$$

TABLE 5
Virtual Channel Selection for the Example Network in Fig. 2

	Source IP	Input of SW_1	Input of SW_2	Input of SW_3	Input of SW_4	Destination IP
Flow 1	Vch-1	Vch-1	Vch-1	Vch-1	--	Vch-1
Flow 2	Vch-1	Vch-1	Vch-2	Vch-2	Vch-1	Vch-1
Flow 3	Vch-1	Vch-2	--	--	--	Vch-1
Flow 4	Vch-1	--	--	--	Vch-1	Vch-2

$u_1^0[1] = U_1^0[1]$ $u_1^1[1] = U_1^1[1]$ $u_1^2[1] = U_1^2[1]$ $u_1^3[1] = U_1^3[1]$	$u_2^0[1] = U_2^0[1]$ $u_2^1[1] = U_2^1[2]$ $u_2^2[2] = U_2^2[2]$ $u_2^3[2] = U_2^3[1]$ $u_2^4[1] = U_2^4[1]$
$U_1^0[1] = U_1^1[1] =$ $U_1^2[1] = U_1^3[1] = 2\,L_1$ $UB_1[1] = 8\,L_1, MI_1 = 2\,L_1$	$U_2^0[1] = U_2^1[1] = U_2^2[1] = U_2^3[1] = U_2^4[1] = 2\,L_2$ $UB_2 = 10L_2, MI_2 = 2L_2$
$u_3^0[1] = U_3^0[2]$ $u_3^1[2] = U_3^1[1]$	$u_4^0[1] = U_4^0[1]$ $u_4^1[1] = U_4^4[2]$
$U_3^0[1] = U_3^1[1] = 2L_3$ $UB_3 = 4L_3, MI_3 = 2L_3$	$U_4^0[1] = U_4^4[2] = 2L_4$ $UB_4 = 4L_4, MI_4 = 2L_4$

Fig. 7. The calculation of UB_i and MI_i for the example NoC in Fig. 2 using RTB-HB, considering two virtual channels per physical channel.

To illustrate the support of virtual channels, let us consider the example shown in Fig. 2 with two virtual channels per physical channel. In this case, Table 5 shows the mapping between flows and virtual channels at each switch input port. Fig. 7 shows the calculation for this example. By comparing the results with the case where no virtual channels are used, it can be observed that virtual channels can reduce the number of flow contentions in different switches. At the same time, the bandwidth of a physical channel is shared among its virtual channels, thus depending on the application and on the strategy that assigns virtual channels to different flows, the worst case latency and bandwidth values can be better or worse compared to the case without virtual channels, regardless of extra hardware cost and complexity of virtual channels.

4.2 The Proposed Delay Model RTB-LL

We present a substantial improvement to the previously published method WCFC [10]. WCFC also calculates the upper bound propagation delays and permitted injection intervals for the flows in a wormhole network. It considers the arbitration contention packets face and the delay incurred by other packets sharing some part of their route due to such blockings. With a notation similar to that used above, WCFC employs [10] the following equation to calculate UB_i and mI_i:

$$\begin{aligned}
UB_i &= ts_1 + ts_2 + L_i + a + \sum_j u_i^j, \quad j = 0..h_i \\
mI_i &= ts_1 + L_i + \sum_j u_i^j - h_i \times S_d,\ j = 0..h_i \\
u_i^j &= S_d + \sum_x U_{I(x)}^j, x = 1..\ z_c(i,j),\ 0 \leq j \leq h_i.
\end{aligned} \tag{18}$$

In the WCFC method, the calculations are based on the assumption that each flow injects packets with a minimum permitted interval. For the applications that can support such an assumption, we present a method that provides significant improvement in bound tightness over the WCFC method, which we call RTB-LL. As RTB-LL is less

$u_1^0 = 0$ $u_1^1 = S_d + U_2^1$ $u_1^2 = S_d + U_2^2$ $u_1^3 = S_d$ $U_2^1 = U_2^2 + U_1^2$ $U_2^2 = U_2^3 = U_2^4 + U_4^4 = L_2 + L_3$ $U_1^2 = U_1^3 = L_1$ $UB_1 = a + 3S_d + L_1 + 2L_2 + 2L_4$ $mI_1 = 2L_1 + 2L_2 + 2L_4$	$u_2^0 = U_3^0$ $u_2^1 = S_d + U_1^1$ $u_2^2 = S_d + U_1^2$ $u_2^3 = S_d$ $u_2^4 = S_d + U_4^4$ $U_3^0 = U_3^1 = L_3$ $U_1^1 = U_1^2 + U_2^2$ $UB_2 = a + 4S_d + 2L_1 + 2L_2 + L_3 + 2L_4$ $mI_2 = 2L_1 + 2L_2 + L_3 + 2L_4$
$u_3^0 = U_2^0, u_3^1 = S_d$ $U_2^0 = U_2^1 + U_1^1$ $UB_3 = a + S_d + 2L_1 + 2L_2 + L_3 + 2L_4$ $mI_3 = 2L_1 + 2L_2 + L_3 + 2L_4$	$u_4^0 = 0$ $u_4^4 = S_d + U_2^4$ $UB_4 = a + S_d + L_2 + L_4$ $mI_4 = L_2 + L_4$

Fig. 8. The complete calculation of UB_i and mI_i for the example NoC of Fig. 2 with method WCFC.

$u_1^0 = 0$ $u_1^1 = S_d + U_2^1$ $u_1^2 = S_d$ $u_1^3 = S_d$ $U_2^1 = U_2^2 = U_2^3 = U_2^4 + U_4^4 = L_2 + L_4$ $U_1^2 = U_1^3 = L_1$ $UB_1 = a + 3S_d + L_1 + L_2 + L_4$ $mI_1 = L_1 + L_2 + L_4$	$u_2^0 = U_3^0$ $u_2^1 = S_d + U_1^1$ $u_2^2 = S_d$ $u_2^3 = S_d$ $u_2^4 = S_d + U_4^4$ $U_3^0 = U_3^1 = L_3, U_1^1 = U_1^2$ $UB_2 = a + 4S_d + L_1 + L_2 + L_3 + L_4$ $mI_2 = L_1 + L_2 + L_3 + L_4$
$u_3^0 = U_2^0, u_3^1 = S_d$ $U_2^0 = U_2^1 + U_1^1$ $UB_3 = a + S_d + L_1 + L_2 + L_3 + L_4$ $mI_3 = L_1 + L_2 + L_3 + L_4$	$u_4^0 = 0$ $u_4^4 = S_d + U_2^4$ $UB_4 = a + S_d + L_2 + L_4$ $mI_4 = L_2 + L_4$

Fig. 9. The complete calculation of UB_i and mI_i for the example NoC of Fig. 2 with method RTB-LL.

pessimistic than WCFC in evaluating worst case performance, it enables the design of more hardware-efficient NoCs. To improve upon WCFC, a new concept, called *overlapping flows*, is introduced. If two or more different flows contend for the same output port at a switch, and they also share the same input port, we call such flows *overlapping* at the switch. This notion allows us to significantly optimize the bound tightness.

When F_i contends with multiple overlapping flows at a switch, it is possible to locally coalesce all such overlapping flows into a single one. This is because the arbitration cannot be lost to multiple of those flows, as they cannot physically produce a contending packet simultaneously given that they enter the switch through the same input port. If there exist, e.g., two overlapping contending flows at hop j with delay parameters U_{i1}^j and U_{i2}^j, it is possible to consider $MAX(U_{i1}^j, U_{i2}^j)$ as their representative delay, instead of $U_{i1}^j + U_{i2}^j$, for calculating the parameters of F_i. Moreover, whenever F_i overlaps with other flows, those other contending flows should be ignored. By applying these optimizations, we have noticed a significant improvement in bound tightness for RTB-LL, shown in the next section and as summarized in Table 1.

Figs. 8 and 9 show the calculated UB_i and mI_i values for both WCFC and RTB-LL methods for the same example in Fig. 2. Since flows F_1 and F_2 are overlapping at SW_2, our proposed RTB-LL improves the bound tightness compared to WCFC. Consider, for the sake of exemplification, a NoC variant shown in Fig. 10, with another contending flow at SW_2. In RTB-LL, the delay u_3^2 of F_3 at SW_2 can be modeled as $S_d + MAX(U_1^2, U_2^2)$ instead of the overly pessimistic value $S_d + U_1^2 + U_2^2$ calculated by WCFC.

To show how virtual channels are supported in RTB-LL, the same example for method RTB-HB is considered. Fig. 11

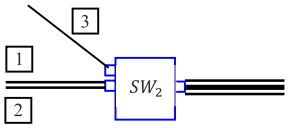


Fig. 10. NoC variant where flow F_3 contends with two overlapping flows F_1 and F_2 at SW_2.

$u_1^0[1] = 0$ $u_1^1[1] = S_d$ $u_1^2[1] = S_d$ $u_1^3[1] = S_d$ $UB_1 = a + 3S_d + 2L_1$ $mI_1 = 2L_1$	$u_2^0[1] = 0$ $u_2^1[1] = S_d$ $u_2^2[2] = S_d$ $u_2^3[2] = S_d$ $u_2^4[1] = S_d$ $UB_2 = a + 4S_d + 2L_2, mI_2 = 2L_2$
$u_3^0[1] = 0$ $u_3^1[2] = S_d$ $UB_3 = a + S_d + 2L_3$ $mI_3 = 2L_3$	$u_4^0[1] = 0$ $u_4^4[1] = S_d$ $UB_4 = a + S_d + 2L_4$ $mI_4 = 2L_4$

Fig. 11. Calculation of UB_i and mI_i for the example in Fig. 2 using RTB-LL and WCFC, with two virtual channels per physical channel.

TABLE 6
Network Parameters for the Study

L	a	b_1	b'_1	b_2	b_3	b'_3	B_d	S_d	ts_1	ts_2	$FlitWidth$	$Freq$
flits	*regs*	*regs*	*regs*	*regs*	*regs*	*regs*	*regs*	*regs*	*cycles*	*cycles*	*bytes*	*MHz*
4	1	1	1	2	0	0	4	4	0	0	4	400

shows the results. Like in RTB-HB, using virtual channels results in better latency and bandwidth bounds, since flow contentions are resolved.

5 Studies on Applications

The proposed methods RTB-HB and RTB-LL can be used to analyze the scheduling of traffic flows in real-world applications. In this section, we present studies on a multimedia application (four other multimedia and RT applications are considered in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2011.240). We compare methods RTB-HB and RTB-LL to the baseline method WCFC, using the parameters listed in Table 6. In these applications, we assume that NoC topologies are predefined based on application communication requirements, but without any feedback from the proposed algorithms to customize the network structure for better upper bound delay and interval time results (considering such a feedback is a possible extension for future work). In particular, for many applications, it is possible to identify a small subset of flows as critical, and then to optimize the NoC based on feedback loops from RTB-HB and RTB-LL to improve the performance of such critical flows. It is possible to do this without dedicated hardware support or any priority scheme.

5.1 Case Study: A Multimedia Application

In this section, we compare the results of applying RTB-HB, RTB-LL, and WCFC to D26-media (Fig. 12), a real-time multimedia application with 67 communication flows, some of which critical. The application is mapped onto different NoC topologies, each with different switch counts and switch radices. Fig. 14 shows the average flow latency for different analysis methods and topologies. In particular we have shown the implementation for switch counts up to

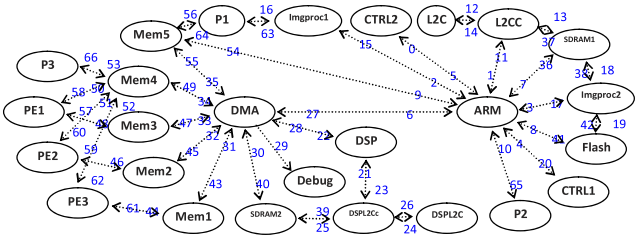


Fig. 12. Communication graph for D26-media.

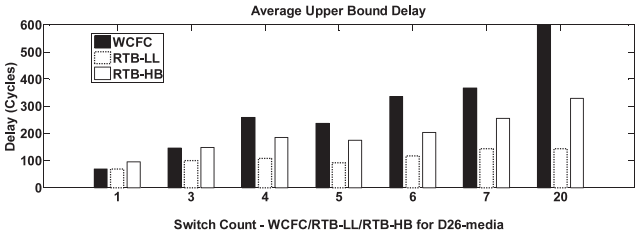


Fig. 14. Average upper bound delay of the flows in D26-media for methods WCFC, RTB-LL, and RTB-HB, applied to topologies with different switch counts.

seven that are typical values and a topology with 20 switches that is a reasonable point with many longer hop flows. Topologies with one or few switches (e.g., 1-3 switches in this example) need them to be "fat" (high-radix), while other cases need "medium" or "thin" (low-radix) switches. It is important to note that the limitations in physical implementation (like power consumption, area, frequency, etc.) may limit the use of fat switches in practice; we still present the results for these cases for the sake of latency comparison, and assume that proper constraints can be implemented at a higher level in the NoC synthesis flow. Two detailed implementations with 5 and 20 switches are shown in Fig. 13. Fig. 15 presents the results in terms of latency, intervals and bandwidth for the whole set of flows for 1-, 5- and 20-switch networks. Figs. 15a, 15b, and 15c compare UB_i. The RTB-LL model always provides the tightest bounds. Compared to WCFC, the largely improved tightness (more than 50 percent on average) is due to the analysis of overlapping flows, a novelty of this paper. Please note that the improved tightness comes without any impact on the accuracy of the bounds, which are still under

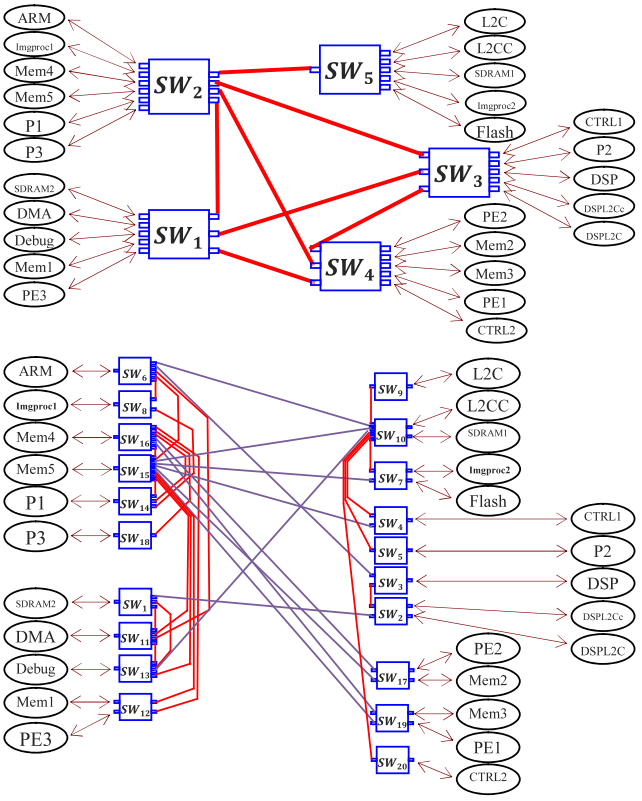


Fig. 13. D26-media application mapped on 5-switch and 20-switch NoC.

worst case assumptions. For the topology with only one switch and without overlapping flows in Fig. 15a, the results for RTB-LL and WCFC are identical but increasing switch counts triggers different performance profiles. RTB-HB is intrinsically expected to return higher worst case latencies, due to the assumption that no hardware traffic injection regulation facilities are available. Still, due to the more accurate calculation approach, the bounds are on average 30 percent lower than those given by WCFC, despite the less restrictive assumptions. There are, however, a few flows for which WCFC predicts lower delays than RTB-HB, due to the regulated injection assumption. In a zero-load scenario (with no contention at all), the minimum theoretical latency to traverse the 5-switch NoC for flows spanning a single hop is eight cycles $(S_d + L)$, while RTB-LL gives a minimum upper-bound of 17 cycles in worst case contention. The delays calculated for the 20-switch topology, in this example, are higher, as a result of longer paths (more hops) per flow, higher probability of contention, and especially for RTB-HB, more in-flight packets. This suggests, as intuitively expected, that NoCs with fewer hops guarantee, on average, lower delay bounds. As described earlier, physical implementation limitations may prevent using fat switches in practice. On the other hand, increasing the number of switches not only increases the system cost but requires a careful consideration in the design process to reduce flow contentions to acquire tighter upper bound delays; so, a trade off may be considered for the number of switches. For RTB-LL, the number of contending in-flight packets is unrelated with the NoC topology, so the delay will not increase as a result of more hops.

Figs. 15d, 15e, and 15f show the maximum and minimum injection intervals (MI_i and mI_i). Intuitively, if traversal delays are lower, new packets can be injected sooner, so MI_i (mI_i) plots resemble UB_i trends: flows with lower latencies can be injected more frequently. Thus, the mI_i intervals are always shorter in RTB-LL and the MI_i intervals often shorter in RTB-HB when compared to mI_i in WCFC (except when using a few fat switches). These intervals can be directly translated into mBW_i and MBW_i using the equations in Table 3. Results are shown in Figs. 15g, 15h, and 15i. The maximum injectable bandwidths (MBW_i) are, on average 35 percent higher according to RTB-LL when compared to WCFC, and 25 percent higher according to the minimum bandwidth (mBW_i) in RTB-HB. The maximum theoretical injectable bandwidth is 1,600 MB/s (Freq $\times$ FlitWidth); according to RTB-LL, even under worst case assumptions, some flows on the 5-switch NoC are guaranteed injection

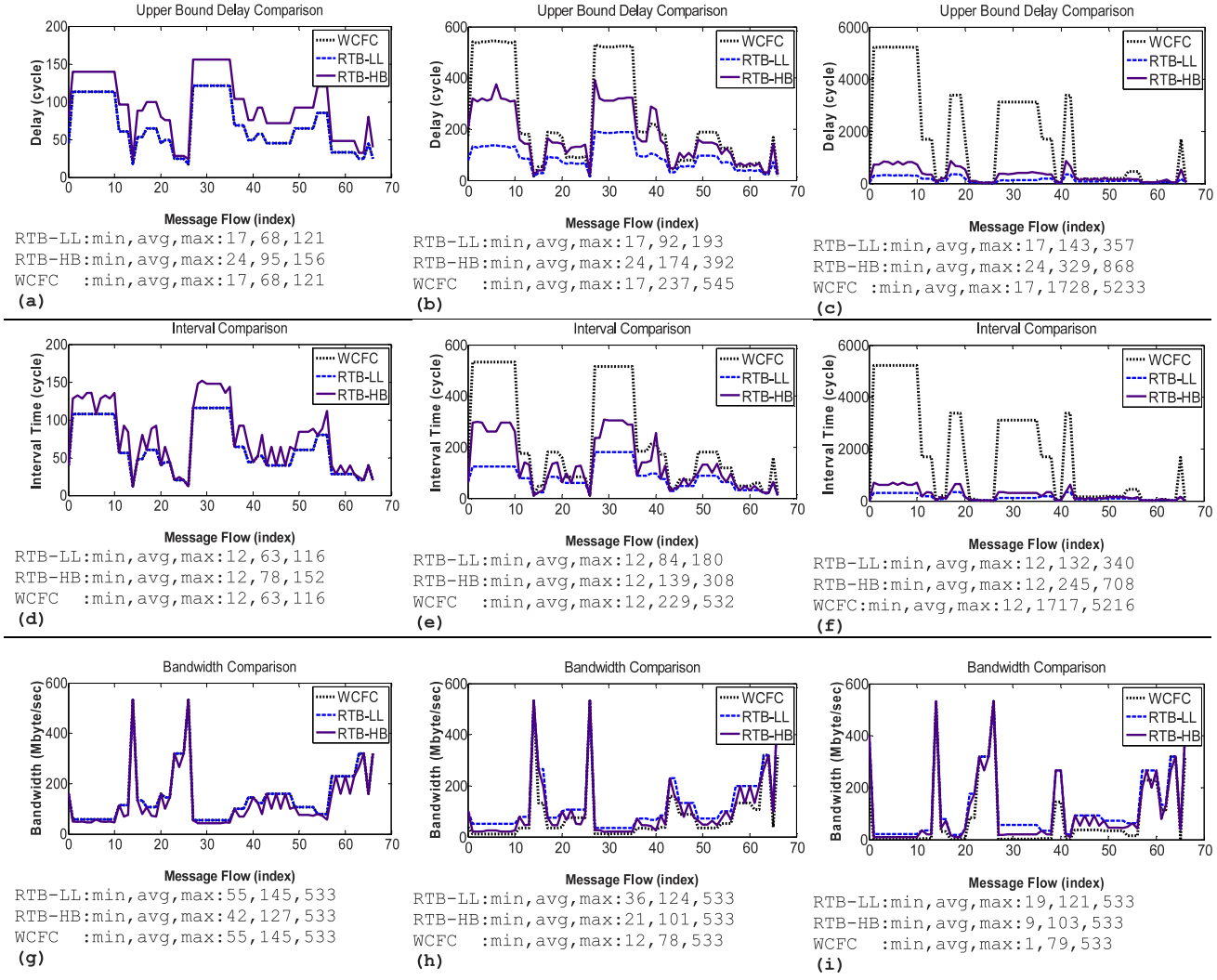


Fig. 15. (a), (d), (g) $UB_i, MI_i(mI_i)$ and $mBW_i(MBW_i)$ characterized with WCFC, RTB-LL, and RTB-HB for D26-media mapped onto a 1-switch NoC. (b), (e), (h) The same metrics mapping onto a 5-switch NoC. (c), (f), (i) mapping onto a 20-switch NoC. Horizontal axes enumerate the communication flows.

rates of as much as 533 MB/s. In the 20-switch network, the higher contention likelihood affects injectable bandwidth negatively, but the use of more resources has a positive effect on many-hop flows, resulting in comparable injectable bandwidths. In summary, NoCs with few hops exhibit clearly better flow average upper bound traversal delays, but in terms of injectable bandwidths, the mapping of the flows (i.e., the contention patterns) and the amount of used resources play a decisive performance role.

5.2 The Effect of Virtual Channels

The results of employing different number of virtual channels in the network in Fig. 12 are reported in Figs. 16 and 17. Here we consider D26-media application with RTB-HB and RTB-LL methods for one, two and four virtual channels per physical channel. The strategy that assigns virtual channels to the flows is to share the load of input ports among the virtual channels and thus minimizing the contentions on switch output ports. The figures show that

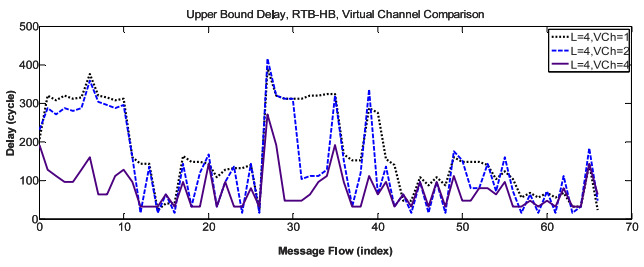


Fig. 16. Worst case delay comparison for D26-media application, in method RTB-HB with 1, 2, and 4 virtual channels.

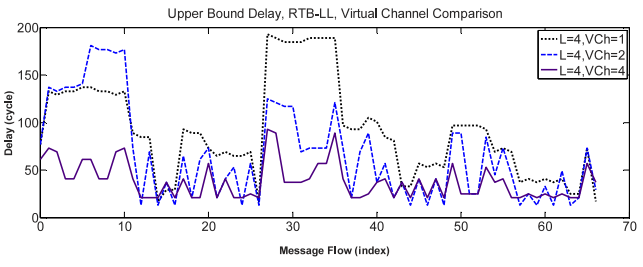


Fig. 17. Worst case delay comparison for D26-media application, in method RTB-LL with 1, 2, and 4 virtual channels.

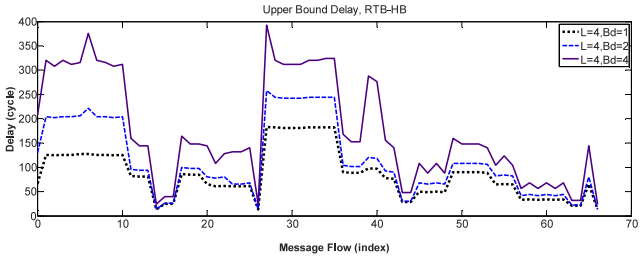


Fig. 18. Worst case delay comparison for D26-media application, $L = 4$, and $B_d = 1, 2,$ and 4.

for both methods increasing the number of virtual channels results in better average RT metrics for different flows.

5.3 Study with Variable Buffer Depths

Figs. 18 and 19 show the comparison of worst case delay and maximum interval for D26-media application for the case $B_d < L$. The figure suggests that shallower buffers, when the message length is fixed, will result in smaller worst case delay and maximum interval bounds. Fig. 20 illustrates the effects of increasing the buffer depth B_d, in all switches of the NoC, as an integer multiple of message length L. The test is run for the D26-media application using method RTB-HB. A linear relationship can be observed between B_d and the upper bound delay UB_i. But because of the pipeline effect, as described in Section 4.1.2, there is no such a relation between B_d and mBW_i.

A contradiction may be perceived since deeper buffers are generally expected to improve performance, while Fig. 20a reveals worse latency with deeper buffers. In fact, the *average* latency is probably improved with a larger B_d , but worst case latency is not, as shown in Fig. 20a. To understand why the worst case latency deteriorates, consider the following explanation. When the basic case of $B_d = L$ is considered, RTB-HB calculates the maximum delay for a packet P_i from the time the packet is supposed to be injected into the network until when it is ejected at the destination. Since the output buffer of the source core has a depth of $B_d = L$, it is not possible to have more than one packet in this buffer awaiting to be serviced before P_i. In the worst situation, the traffic generator can inject a packet every WI_i cycles, at most; if it injects more, the traffic generator may be stalled by NoC backpressure until this interval elapses. In the more complex case $B_d > L$, some packets may be queued in the source core buffer ahead of P_i, and since RTB-HB imposes no restriction on injection rates, they are expected to be there by a worst case analysis. Thus, the calculated worst case delay for P_i includes the extra time needed to service them. Exactly the same reasoning applies to $B_d > L$ in other intermediate buffers in the network. As a consequence, deeper buffers increase worst case latencies, but this is not incompatible with the fact that increasing buffer size will decrease the *average delay*. Indeed if we apply the same traffic pattern to two identical networks, but with different buffer sizes, the total average delay in the network with larger buffers will typically be lower than that of the other network. Our method RTB-LL does not consider the buffering space B_d to calculate the worst case delay; instead it uses stage delay S_d. Therefore, changing the buffer size will not affect the calculated worst case delay.

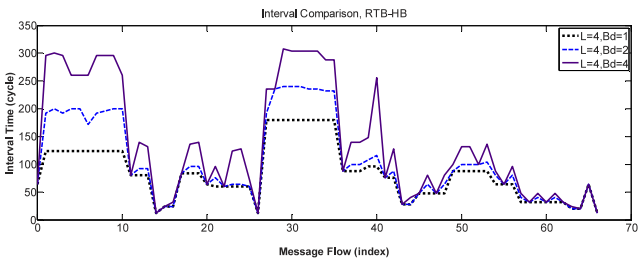


Fig. 19. Maximum interval comparison for D26-media application, $L = 4$, and $B_d = 1, 2,$ and 4.

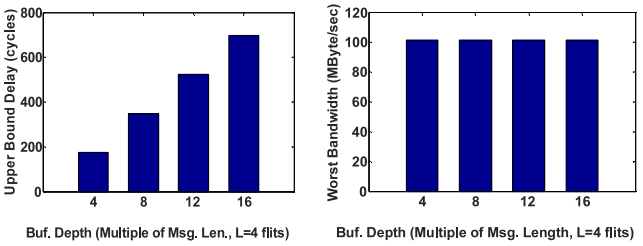


Fig. 20. (a) Average flow Upper Bound Delay versus Buffer Depth on left and (b) Average flow Bandwidth versus Buffer Depth when it is a multiple of Message Length on right in D26-media mapped on 5-switch NoC.

5.4 Suitability to Critical Flows

Fig. 21 shows the average UB_i traversal delay and the average mBW_i injectable bandwidth for the flows traversing x hops of the 5-switch NoC, considering the D26-media application and using RTB-HB. It is seen 1-hop flows exhibit reasonably low latencies and high bandwidths, suitable for critical traffic loads. Thus, the proposed methodology has a clear applicability to industrial RT applications.

6 Complexity of the Methods

To estimate the time complexity of the proposed methods, we calculate the maximum number of required operations. As (1), (3) and (5) show, the only operations are additions and comparisons (for the *MAX* operator); we consider one cycle to execute each of such operations. We call h the maximum number of switches traversed by a flow, and k the number of flows. We also pessimistically assume the maximum number of contending flows at a switch output to be k. For calculating one U_i^j parameter, we need (according to (5)) at most k comparisons and k additions, thus a total of $2k$ operations. The number of U_i^j parameters to be calculated is hk; so, the maximum number of

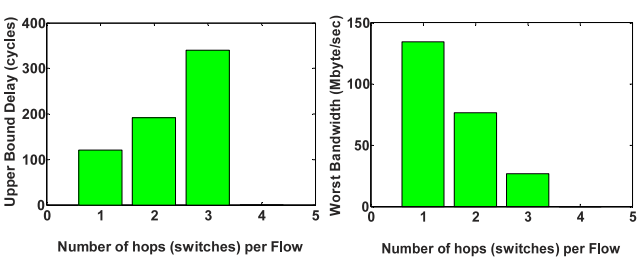


Fig. 21. (a) Average Upper Bound Delay on left and (b) Average Minimum Bandwidth for x-hop flows on right in D26-media for RTB-HB Method.

operations is $2hk^2$. Each u_i^j parameter (except for $j = 0$) can be derived from the equality $U_i^j = u_i^j + 1$; thus, we only need to calculate the case of u_i^0 for all flows. In this case, one u_i^0 (3) needs $2k$ operations; for all u_i^0 parameters we need $2k^2$ operations.

In RTB-HB, the outcome is $k \times UB_i$ and $k \times MI_i$ values. For calculating one UB_i value (according to (1)), we need $h + 1$ additions; so, for all $k \times UB_i$ values, we need $(h + 1)k$ operations, while k operations are needed in the case of MI_i. The total number of operations is the summation of all the above, i.e., $2hk^2 + 2k^2 + (h + 1)k + k$. Therefore, the complexity of the algorithm is $O(hk^2)$.

For RTB-LL, using the same approach, we can show that the complexity of the algorithm for calculating UB_i and mI_i is again $O(hk^2)$. Thus, both algorithms have quadratic time complexity. Also the timing complexity of WCFC algorithm is similar to RTB-LL as it exhibits a similar recursive behavior [10]. In practice, the execution time for all our test applications is very small (few seconds on a standard PC) and the modeling of delay and bandwidth parameters does not pose significant runtime issues.

7 Conclusion and Future Work

We have proposed two different methods to characterize bandwidth and latency for NoC-based real-time SoCs, aiming at guaranteed QoS provisions. The choice of the most suitable method depends on the performance demands of the system and on whether dedicated hardware facilities can be provided by the NoC. One method is aimed at applications demanding the minimum latencies and requires injection regulation, while the other is suitable for applications where packet injection must be flexible to accommodate for higher average injected bandwidths and no hardware regulation is possible. We have proved that the proposed methods return the worst case metrics in a much tighter way than existing approaches, rendering them quite applicable for real-world SoC applications. The next step is to use the results of this work as an input to NoC synthesize and optimization tools whereby the QoS demands of critical traffic flows are met.

Acknowledgments

This work is partly supported by EU ARTIST-DESIGN and ARTEMIS project SMECY.

References

[1] W.J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. 38th Ann. Design Automation Conf. (DAC),* pp. 684-689, 2001.

[2] L. Benini and G.D. Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer J.,* vol 35, no. 1, pp. 70-78, Jan. 2002.

[3] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE '00),* pp. 250-256, 2000.

[4] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Network Delays and Link Capacities in Application-Specific Wormhole NoCs," *Proc. 17th Int'l Conf. VLSI Design,* May 2007.

[5] J. Henkel, W. Wolf, and S. Chakradhar, "On-Chip Networks: A Scalable, Communication-Centric Embedded System Design Paradigm," *Proc. 17th Int'l Conf. VLSI Design (VLSID '04),* vol. 17, pp. 845-851, Jan. 2004.

[6] S. Furber and J. Bainbridge, "Future Trends in SoC Interconnect," *Proc. IEEE Int'l Symp. VLSI Design, Automation and Test,* pp. 183-186, 2005.

[7] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," *Proc. ACM/IEEE Second Int'l Symp. Networks-on-Chip,* pp. 161-170, 2008.

[8] C. Paukovits and H. Kopetz, "Concepts of Switching in the Time-Triggered Network-on-Chip," *Proc. IEEE Int'l Conf. Embedded and Real-Time Computing Systems,* pp.120-129, 2008.

[9] K. Goossens, J. Dielissen, and A. Radulescu, "The Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design and Test of Computers,* vol. 22, no. 5, pp. 414-421, Sept./Oct. 2005.

[10] S. Lee, "Real-Time Wormhole Channels," *J. Parallel Distributed Computer,* vol. 63, pp. 299-311, 2003.

[11] D. Rahmati et al., "A Method for Calculating Hard QoS Guarantees for Networks-on-Chip," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design, ICCAD,* pp. 579-586, 2009.

[12] D. Kandlur, K. Shin, and D. Ferrari, "Real-Time Communication in Multihop Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 10, pp. 1044-1056, Oct. 1994.

[13] M. Zhang, J. Shi, T. Zhang, and Y. Hu, "Hard Real-Time Communication over Multi-Hop Switched Ethernet," *Proc. Int'l Conf. Networking, Architecture, and Storage,* pp. 121-128, 2008.

[14] S. Gopalakrishnan, S. Lui, and M. Caccamo, "Hard Real-Time Communication in Bus-Based Networks," *Proc. IEEE 25th Int'l Real-Time Systems Symp.,* pp. 405-414, 2004.

[15] A. Yiming and T. Eisaka, "A Switched Ethernet Protocol for Hard Real-Time Embedded System Applications," *Proc. 19th Conf. Advanced Information Networking and Applications,* pp. 41-44, Mar. 2005.

[16] K. Watson and J. Jasperneite, "Determining End-to-End Delays Using Network Calculus," *Proc. Fifth IFAC Int'l Conf. Fieldbus Systems and Their Applications (IFAC-FET '03),* pp. 255-260, July 2003.

[17] J. Chen, Z. Wang, and Y. Sun, "Real-Time Capability Analysis for Switch Industrial Ethernet Traffic Priority-Based," *Proc. Int'l Conf. Control Applications,* pp. 525-529, Sept. 2002.

[18] J. Jaspernite, P. Neumann, M. Theis, and K. Watson, "Deterministic Real-Time Communication with Switched Ethernet," *Proc. IEEE Fourth Int'l Workshop Factory Comm. Systems (WFCS '02),* pp. 11-18, 2002.

[19] S. Lee, K.C. Lee, and H.H. Kim, "Maximum Communication Delay of a Real-Time Industrial Switched Ethernet with Multiple Switching Hubs," *Proc. IEEE 30th Conf. Industrial Electronics Soc.,* vol. 3, pp. 2327-2332, 2004.

[20] J. Loeser and H. Haertig, "Low-Latency Hard Real-Time Communication over Switched Ethernet," *Proc. 16th Euromicro Conf. Real-Time Systems (ECRTS),* pp. 13-22, 2004.

[21] J. Kiszka, B. Wagner, Y. Zhang, and J. Broenink, "RTnet - A Flexible Hard Real-Time Networking Framework," *Proc. IEEE 10th Int'l Conf. Emerging Technologies and Factory Automation,* pp. 450- 456, 2005.

[22] J. Hu, U.Y. Ogras, and R. Marculescu, "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 12, pp. 2919-2933, Dec. 2006.

[23] A.E. Kiasari, H. Sarbazi-Azad, and M. Ould-Khaoua, "An Accurate Mathematical Performance Model of Adaptive Routing in the Star Graph," *Future Generation Computer Systems,* vol. 24, no. 6, pp. 461-474, 2008.

[24] J. Kim and C.R. Das, "Hypercube Communication Delay with Wormhole Routing," *IEEE Trans. Computers,* vol. 43, no. 7, pp. 806-814, July 1994.

[25] I. Cohen, O. Rottenstreich, and I. Keslassy, "Statistical Approach to Networks-on-Chip," *IEEE Trans. Computers,* vol. 59, no. 6, pp. 748-761, June 2010.

[26] A.E. Kiasari, H. Sarbazi-Azad, and S. Hessabi, "Caspian: A Tunable Performance Model for Multi-Core Systems," *Proc. 14th Int'l Euro-Par Conf. Parallel Processing,* vol. 5168, pp. 100-109, 2008.

[27] U.Y. Ogras and R. Marculescu, "Analytical Router Modeling for Networks-on-Chip Performance Analysis," *Proc. Conf. Design, Automation and Test in Europe (DATE),* pp. 1-6, 2007.

[28] V. Soteriou, H. Wang, L.-S. Peh, "A Statistical Traffic Model for On-Chip Interconnection Networks," *Proc. IEEE Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems,* pp. 104-116, 2006.

[29] W.J. Dally, "Performance Analysis of k-ary n-Cube Interconnection Networks," *IEEE Trans. Computers,* vol. 39, no. 6, pp. 775-785, June 1990.

[30] J. Draper and J. Ghosh, "A Comprehensive Analytical Model for Wormhole Routing in Multicomputer Systems," *J. Parallel and Distributed Computing,* vol. 23, no. 2, pp. 202-214, 1994.

[31] P. Hu and L. Kleinrock, "An Analytical Model for Wormhole Routing with Finite Size Input Buffers," *Proc. 15th Int'l Teletraffic Congress,* pp. 549-560, 1997.

[32] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger, and Y. Hoskote, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 28, no.1, pp. 3-21, Jan. 2009.

[33] S. Foroutan et al., "An Analytical Method for Evaluating Network-on-Chip Performance," *Proc. Conf. Design, Automation and Test in Europe (DATE),* pp. 1629-1632, 2010.

[34] H. Kopetz et al., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach," *IEEE Micro,* vol. 9, no.1, pp. 25-40, Feb. 1989.

[35] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A Scalable, Single-Chip Communications Architecture," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* pp. 37-46, 2000.

[36] M. Millberg, R.T.E. Nilsson, and A. Jantsch, "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," *Proc. Design, Automation and Test in Europe (DATE) Conf. and Exhibition,* pp. 890-895, 2004.

[37] A. Hansson, M. Subburaman, and K. Goossens, "aelite: A Flit-Synchronous Network on Chip with Composable and Predictable Services," *Proc. Design, Automation and Test in Europe (DATE),* pp.250-255, 2009.

[38] J. Diemer and R. Ernst, "Back Suction: Service Guarantees for Latency-Sensitive on-Chip Networks," *Proc. Networks-on-Chip Int'l Symp. (NOCS),* pp. 155-162, 2010.

[39] A. Hansson, M. Wiggers, A. Moonen, K. Goossens, and M. Bekooij, "Enabling Application-Level Performance Guarantees in Network-Based Systems on Chip by Applying Dataflow Analysis," *IET J. Computers and Digital Techniques,* vol. 3, no. 5, pp. 398-412, Sept. 2009.

[40] E. Bolotin et al., "QNoC: QoS Architecture and Design Process for Network on Chip," *J. Systems Architecture,* vol. 50, nos. 2/3, pp. 105-128, 2004.

[41] T. Bjerregaard and J. Sparsoe, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip," *Proc. Design, Automation and Test in Europe (DATE),* vol. 2, pp. 1226-1231, 2005.

[42] A. Bouhraoua and M.E. Elrabaa, "A High-Throughput Network-on-Chip Architecture for Systems-on-Chip Interconnect," *Proc. Intl. Symp. System-on-Chip (Soc),* pp. 1-4, Nov. 2006.

[43] F. Felicijan and S. Furber, "An Asynchronous on-Chip Network Router with Quality-of-Service (QoS) Support," *Proc. IEEE Int'l SOC Conf. (SOCC),* pp. 274-277, Sept. 2004.

[44] N. Kavaldjiev et al., "A Virtual Channel Network-on-Chip for GT and BE Traffic," *Proc. IEEE CS Ann. Symp. Emerging VLSI Technologies and Architectures (ISVLSI),* Mar. 2006.

[45] A. Leroy et al., "Spatial Division Multiplexing: A Novel Approach for Guaranteed Throughput on NoCs," *Proc. IEEE/ACM/IFIP Int'l Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS),* pp. 81-86, 2005.

[46] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Evaluation of Current QoS Mechanisms in Network on chip," *Proc. Int'l Symp. System-on-Chip (Soc),* pp. 115-118, 2006.

[47] R. Mullins, A. West, and S. Moore, "The Design and Implementation of a Low-Latency on-Chip Network," *Proc. Asia and South Pacific Conf. Design Automation (ASP-DAC),* 2006.

[48] A. Radulescu et al., "An Efficient on-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration," *IEEE Trans. Computer-Aided Design,* vol. 24, no. 1, pp. 4-17, Jan. 2005.

[49] E. Rijpkema et al., "Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Network on Chip," *IEE Proc. Computers and Digital Techniques,* vol. 150, no. 5, pp. 294-302, Sept. 2003.

[50] E. Salminen, A. Kulmala, and T. Hamalainen, "Survey of Network-on-Chip Proposals," www.ocpip.org, Mar. 2008.

[51] S. Balakrishnan and F. Ozguner, "A Priority-Driven Flow Control Mechanism for Real-Time Traffic in Multiprocessor Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 7, pp. 665-678, July 1998.

[52] Y. Qian, Z. Lu, and W. Dou, "Analysis of Worst-Case Delay Bounds for Best-Effort Communication in Wormhole Networks on Chip," *IEEE Trans. Computer-Aided Design,* vol. 29, no. 5, pp. 802-815, May 2010.

[53] C. Chang, *Performance Guarantees in Communication Networks.* Springer-Verlag, 2000.

[54] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Springer-Verlag, 2001.

[55] F. Jafari, Z. Lu, A. Jantsch, and M.H. Yaghmaee, "Optimal Regulation of Traffic Flows in Networks-on-Chip," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE),* pp. 1621-1624, 2010.

[56] M. Bakhouya et al., "Analytical Modeling and Evaluation of On-Chip Interconnects Using Network Calculus," *Proc. ACM/IEEE Int'l Symp. Networks-on-Chip (NOCS),* pp. 74-79, 2009.

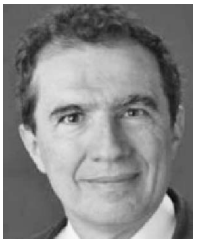
[57] S. Murali et al., "Design of Application-Specific Networks on Chips with Floorplan Information," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD),* pp. 355-362, 2006.

Dara Rahmati received the BSc degree in computer engineering from the University of Tehran, Iran, in 1998, the MSc degree in computer engineering from the University of Tehran in 2001. He is currently working toward the PhD degree in computer engineering at Sharif University of Technology, Tehran, Iran. His research interests are in design and performance evaluation of optimized networks-on-chip architectures including topology selection, routing and switching algorithms, performance boundary evaluation, and QoS support for Real Time networks-on-chip. He is a student member of the IEEE.

Srinivasan Murali received the MS and PhD degrees in electrical engineering from Stanford University, Palo Alto, CA, in 2007. He is a cofounder and the chief technical officer with iNoCs, Lausanne, Switzerland. He is also currently a research scientist with Ecole Polytechnique Federale de Lausanne. He has authored a book and presented more than 40 publications in leading conferences and journals. His current research interests include interconnect design for systems-on-chips, thermal modeling, and reliability of multicore systems. He is a recipient of the European Design and Automation Association Outstanding Dissertation Award in 2007 for his work on interconnect architecture design. He received the Best Paper Award at the Design Automation and Test in Europe Conference in 2005. One of his papers has also been selected as one of "The Most Influential Papers of 10 Years DATE". He is a member of the IEEE and the IEEE Computer Society.

Luca Benini received the PhD degree in electrical engineering from Stanford University in 1997. He is a full professor at the Department of Electrical Engineering and Computer Science (DEIS) of the University of Bologna. He also holds a visiting faculty position at the Ecole Polytechnique Federale de Lausanne (EPFL) and he is currently serving as a chief architect for the Platform 2012 project in STmicroelectronics, Grenoble. His research interests are in energy-efficient system design and multicore SoC design. He is also active in the area of energy-efficient smart sensors and sensor networks for biomedical and ambient intelligence applications. He has published more than 500 papers in peer-reviewed international journals and conferences, four books and several book chapters. He has been general chair and program chair of the Design Automation and Test in Europe Conference. He has been an associate editor of several international journals, including the *IEEE Transactions on Computer Aided Design of Circuits and Systems* and the *ACM Transactions on Embedded Computing Systems*. He is a fellow of the IEEE, a member of the Academia Europaea, and a member of the steering board of the ARTEMISIA European Association on Advanced Research & Technology for Embedded Intelligence and Systems.

Federico Angiolini received the MS degree (summa cum laude) in electrical engineering from the University of Bologna, Bologna, Italy, in 2003, and the PhD degree from the Department of Electronics and Computer Science, University of Bologna, in 2008. He is currently the vice president of Engineering with the INoCs, Lausanne VD, Switzerland. His current research interests include memory hierarchies, multiprocessor-embedded systems, and networks-on-chip. He is a member of the IEEE and the IEEE Computer Society.



Giovanni De Micheli (S'79-M'79-SM'80-F'94) is currently a professor and the director of the Institute of Electrical Engineering and the Integrated Systems Centre, Ecole Polytechnique Federale de Lausanne, Switzerland. He is also a program leader of the Nano-Tera.ch program. He was a professor of electrical engineering at Stanford University. His research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis for emerging technologies, networks on chips 3-D integration, heterogeneous platform design including electrical components and biosensors, as well as in data processing of biomedical information. He is the recipient of the 2003 IEEE Emanuel Piore Award. He is a fellow of the Association for Computing Machinery. He was also the recipient of the Golden Jubilee Medal for outstanding contributions to the IEEE CAS Society in 2000 and the 1987 D. Pederson Award for the best paper on the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD/ICAS). He was the Division 1 Director (2008-2009), cofounder and the president elect of the IEEE Council on EDA (2005-2007), the president of the IEEE CAS Society 2003, the editor in chief of the IEEE TCAD/ICAS (1987-2001). He has been the chair of several conferences, including DATE (2010), pHealth (2006), VLSI SOC (2006), DAC (2000), and ICCD (1989). He is a fellow of the IEEE.

Hamid Sarbazi-Azad received the BSc degree in electrical and computer engineering from Shahid-Beheshti University, Tehran, Iran, in 1992, the MSc degree in computer engineering from Sharif University of Technology, Tehran, in 1994, and the PhD degree in computing science from the University of Glasgow, United Kingdom, in 2002. He is currently a professor of computer engineering at Sharif University of Technology, and heads the School of Computer Science of the Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. His research interests include high performance computer architectures, NoCs and SoCs, parallel and distributed systems, performance modeling/evaluation, graph theory and combinatorics, and wireless/mobile networks, on which he has published more than 200 refereed conference and journal papers. He received Khwarizmi International Award in 2006 and TWAS Young Scientist Award in engineering sciences in 2007. He is a member of the managing board of Computer Society of Iran (CSI), and has been serving as the editor in chief for the CSI *Journal on Computer Science and Engineering* since 2005. He is an associate editor of *IEEE Transactions on Computers* and has guest edited several special issues on high-performance computing architectures and networks (HPCANs) in related journals. He is a member of the ACM and the IEEE.