CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

NoC Contention Analysis using a Branch and Prune Algorithm

Dakshina Dasari
Borislav Nikolic
Vincent Nelis
Stefan M. Petters

CISTER-TR-131107
Version:
Date: 11/13/2013

NoC Contention Analysis using a Branch and Prune Algorithm

Dakshina Dasari, Borislav Nikolic, Vincent Nelis, Stefan M. Petters

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

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

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

Abstract

“Many-core” systems based on a Network-on-Chip (NoC) architecture offer various opportunities in terms of performance and computing capabilities, but at the same time they pose many challenges for the deployment of real-time systems, which must fulfill specific timing requirements at run time – It is therefore essential to identify, at design time, the parameters that have an impact on the execution time of the tasks deployed on these systems and the upper bounds on the other key parameters. The focus of this work is to determine an upper bound on the traversal time of a packet when it is transmitted over the NoC infrastructure. Towards this aim, we first identify and explore some limitations in the existing recursive-calculus based approaches to compute the worst-case traversal time (WCTT) of a packet. Then, we extend the existing model by integrating the characteristics of the tasks that generate the packets. For this extended model, we propose an algorithm called “Branch and Prune” (BP). Our proposed method provides tighter and safe estimates than the existing recursive-calculus based approaches. Finally, we introduce a more general approach - “Branch, Prune and Collapse” (BPC) which offers a configurable parameter that provides a flexible trade-off between the computational complexity and the tightness of the computed estimate. The recursive-calculus methodsand BP present two special cases of BPC when a trade-off parameter is 1 or 1, respectively. Through simulations, we analyze this trade-off, reason about the implications of certain choices and also provide some case studies to observe the impact of task parameters on the WCTT estimates.

NoC Contention Analysis using a Branch and Prune Algorithm

A

DAKSHINA DASARI, BORISLAV NIKOLIĆ, VINCENT NÉLIS, STEFAN M. PETTERS,
CISTER-ISEP Research Centre, Polytechnic Institute of Porto

"Many-core" systems based on a Network-on-Chip (NoC) architecture offer various opportunities in terms of performance and computing capabilities, but at the same time they pose many challenges for the deployment of real-time systems, which must fulfill specific timing requirements at run time – It is therefore essential to identify, at design time, the parameters that have an impact on the execution time of the tasks deployed on these systems and the upper bounds on the other key parameters. The focus of this work is to determine an upper bound on the *traversal time* of a packet when it is transmitted over the NoC infrastructure. Towards this aim, we first identify and explore some limitations in the existing recursive-calculus based approaches to compute the worst-case traversal time (WCTT) of a packet. Then, we extend the existing model by integrating the characteristics of the tasks that generate the packets. For this extended model, we propose an algorithm called "Branch and Prune" (BP). Our proposed method provides tighter and safe estimates than the existing recursive-calculus based approaches. Finally, we introduce a more general approach - "Branch, Prune and Collapse" (BPC) which offers a configurable parameter that provides a flexible trade-off between the computational complexity and the tightness of the computed estimate. The recursive-calculus methods and BP present two special cases of BPC when a trade-off parameter is 1 or ∞, respectively. Through simulations, we analyze this trade-off, reason about the implications of certain choices and also provide some case studies to observe the impact of task parameters on the WCTT estimates.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special Purpose and Application Based Systems—*Real-time and embedded systems*; C.3 [**Computer Systems Organization**]: Multiple Data Stream Architectures—*Interconnection Architectures*; B.4 [**Input/Output and Data Communications**]: Performance Analysis and Design Aids—*Worst-case Analysis*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Many-core systems, Network on-chip, Real-time systems, Wormhole routing

1. INTRODUCTION

The current trend in the chip manufacturing industry is towards the integration of previously isolated functionalities into a single-chip. Following this trend, the usage of multi-cores has become ubiquitous, not only for general-purpose systems but also in the embedded computing segment. Besides the increasing processing demand, advancements in the semiconductor arena have fostered in the "many-core" systems era and we are now witnessing the emergence of chips enclosing up to 100 cores. The explosion of the number of cores within a single chip also ushered in many issues and challenges – system designers realized that the traditional shared bus/ring architecture (c.f. left plot of Figure 1) does not scale beyond a limited number of cores as it results in a substantial increase in the access time to the off-chip subsystems due to contention for the bus. Hence, the increase in the number of cores necessitated a shift in the earlier design paradigm towards a more scalable interconnection medium: the NoC architecture [Benini and De Micheli 2002]. One of the base principles of the many-core technology is the division of the processing elements into "tiles" interconnected by a NoC. Each tile is composed of a processor core, a private cache subsystem and a network switch connected to its (up to 4) neighboring tiles located in the cardinal di-

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

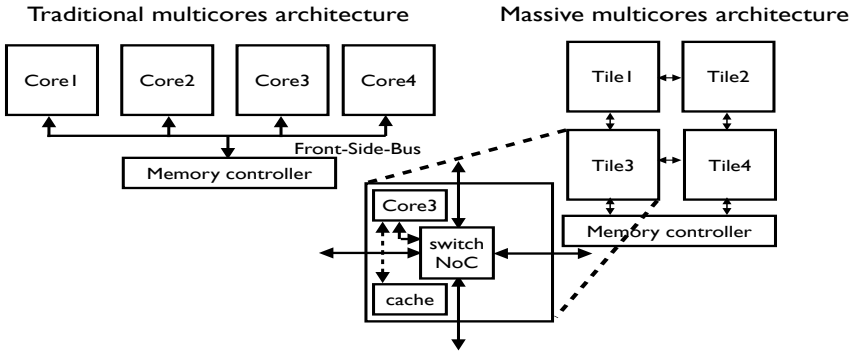


Fig. 1. Multi-core vs. many-core systems

rections, thereby forming a 2D-mesh (c.f. right plot of Figure 1). The NoC serves as a communication channel among the cores and between the cores and other off-chip subsystems, e.g. the main memory. Such many-core systems offer evident enhanced computational capabilities compared to the former (traditional) multi-core platforms. The Tile64 from Tilera [Wentzlaff et al. 2007], Epiphany from Adapteva and the 48-core Single-Chip-Cloud computer [Intel 2010] are just some examples of such many-core architectures. Without loss of generality, in this document we focus on the structure and terminology of the Tile64 platform.

Motivation: Real-time applications and NoCs. NoC-based multicores are the preferred platforms for the deployment of real-time applications, where-in it is very important for applications to exhibit the required functional behavior within pre-defined time-bounds to be deemed correct. It is also vital to *derive upper bounds on the execution time of these applications at design time itself* before these applications can be deployed. In a scenario involving data transfers (amongst cores or from cores to memory), the execution time of a task running on a given core increases as the core stalls waiting for the data to be transferred over the underlying network. This waiting time can lead to a substantial increase in the execution time when the traffic on the network and thus the contention for the network resources increases. Specifically, when the task under analysis is required to meet *some strict timing requirements*, an upper bound for this extra delay must be determined. Additionally, depending on their respective behavior, tasks running on different cores may release packets over the network independently and asynchronously. All the packets are transmitted over the same underlying interconnection network and share the available network resources like links and finite sized buffers. Thus the time to transmit a packet depends on the current load of the network, which is in turn determined by the number of packets generated by the tasks executing on the other cores. Other factors like the routing mechanism employed also impacts the traversal times as it influences the path taken by the packets to reach their destination –this in-turn decides whether they would directly or indirectly block the analyzed packet by contending for the same resources. To summarize, the number of parameters contributing to the non-determinism combined with the large number of cores poses a challenging problem to designers aiming to determine an upper bound on the traversal time of a (message/memory/IO) packet . In this work, we aim to compute such an upper bound which we refer to as the worst-case traversal time (WCTT) for a NoC based many-core system employing a wormhole switching technique [Dally 1992].

2. RELATED WORK

A significant amount of research has been carried out on exploring the impact of the interconnect networks in systems employing wormhole switching [Dally and Seitz 1986]. In the works of [Draper and Ghosh 1994] and [Dally 1992], the respective authors elaborated on the estimation of end-to-end delays for wormhole switching networks, but with the primary focus on the determination of the average latencies using queuing

theory techniques. As mentioned earlier, in real-time systems, estimation of the worst case latencies rather than average case latencies is vital. Hence the earlier approaches do not suffice to perform a real-time analysis.

To ensure predictability and derive upper bounds on the communication delays, some researchers have used mechanisms which require special hardware support to the NoC as in [Diemer and Ernst 2010], priority mechanisms [Shi and Burns 2008], time-triggered systems [Paukovits and Kopetz 2008] and time division multiple access (TDMA) [Goossens et al. 2005]. All these approaches assume that the basic NoC is designed to support predictability, but as seen in a survey of NoCs [Salminen et al. 2008], existing commercial of the shelf (COTS) based NoC architectures are more suited to provide best effort service and hence to model the existing systems, a software-based analysis is warranted.

The existing works which address the issues of the worst-case end-to-end communication latencies in standard NoC-based many-cores can be broadly categorized into two groups: approaches applying network calculus and approaches applying Recursive calculus (RC). We borrowed the terminology RC from [Ferrandiz et al. 2012].

Network Calculus (NC) based methods: In general queuing networks, network calculus-[Boudec and Thiran 2004] provides an elegant way to express and deal with the timing properties of traffic flows. Based on the powerful abstraction of arrival curves for traffic flows and service curves for network elements like routers and servers, it facilitates the computation of the worst-case delay and backlog bounds. For wormhole switching based networks, flow control is based on feedback received from the next router (downstream router). Determining the service curve of a given router independently (without the knowledge of the service curve of the next router involved in the transfer) is not straightforward by the basic abstractions provided by network calculus theory (which is designed to deal with forward networks) since there is a cyclic dependency between the service curves of the routers involved in the transfer.

To overcome this, researchers have modeled the flow control mechanism in the switch itself as another service curve [Qian et al. 2010]. But [Ferrandiz et al. 2011] clearly showed with an illustrative example the flaws of the design and that such a modeling is pessimistic, leading to over-dimensioning of resources. [Ferrandiz et al. 2011] consider a space wire network topology and introduced a special network element called the “wormhole section” to describe the wormhole routing with the network-calculus terminology. This element envelopes a set of routers lying in the shared path between an analyzed flow and the blocking flow(s): the analyzed and the blocking flows enter the first router and exit through the last router of the wormhole section, with no additional blocking flows either entering or leaving from any other link within the wormhole section. The analysis treats this section as a single element, with the arrival and service curves computed as a function of the individual curves of the flows contained within the section. Finally, an end-to-end service curve is derived by combining the service curves of all the wormhole sections in the path of the analyzed flow. In the presence of diverse traffic (with intersecting blocking flows with short shared paths), the direct application of this method on the NoC-based many-core platform would force a wormhole section fragmentation, i.e. every router would be treated as an individual element, which renders the purpose of the wormhole section obsolete, and the results pessimistic.

Recursive Calculus (RC) based methods: The methods centered around this paradigm compute the end-to-end delays by recursively analyzing the contention at each router in the path of the analyzed flow. The model and assumptions to support this methodology have, as a common denominator, the assumption that flows inject packets at the maximum rate to saturate the network. The initial assumption of these approaches is that all the intermediate buffers in the switches between the source and the destination are filled to their capacity [Ferrandiz et al. 2012]. The method thus ensures capturing the worst-case scenario.

The works of [Lee 2003], [Rahmati et al. 2009] and [Ferrandiz et al. 2009] have been noteworthy in this area. Initially, [Lee 2003] proposed a model for real-time communi-

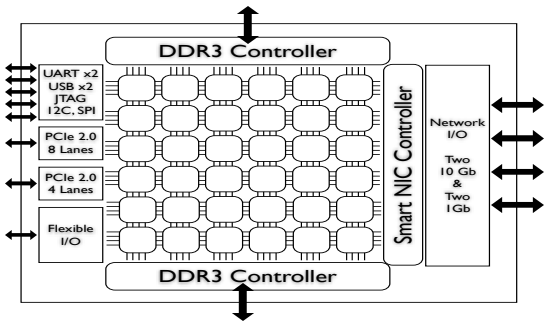


Fig. 2. Tilera architecture. (Diagram taken from [Wentzlaff et al. 2007])

cation in wormhole networks based on the use of real-time wormhole channels. This was improved by [Rahmati et al. 2009] by computing real-time bounds for high bandwidth traffic in which they assume that all intermediate buffers are full, and for low latency regulated traffic, the concept of lumping flows is combined with the method of [Ferrandiz et al. 2009]. [Lu et al. 2005] proposed a contention tree based approach focused at feasibility analysis for a set of periodic messages with pre-assigned priorities, which were used to resolve arbitration conflicts in the switch. Their model does not classically fall into the recursive calculus based methods but it introduced the concept of contention trees (to capture direct and indirect blockings) which are analyzed in a recursive manner and thus conceptually fits in this category (and not in the NC based approaches).

In the approach proposed by [Ferrandiz et al. 2009], which is conceptually similar to the method of [Rahmati et al. 2009], an upper-bound on the traversal delay is computed, but with the assumption that the packets can be injected into the network continuously and therefore the computed WCTT is not tight.

The key advantage of these methods is that they compute the WCTT (with low time complexity), but the main limitation is that they do not leverage the input arrival patterns and hence lead to over-approximations of the WCTT. This may lead to a system design in which the processor utilization is over-dimensioned and many potential tasks which could be scheduled at run-time are deemed unschedulable. Hence a method which provides tighter WCTTs in acceptable times is warranted.

Positioning our approach: In a very recent work by [Ferrandiz et al. 2012], the authors compare their newly proposed NC method against their previous RC method considering different parameters. In our approach we combine the best of both methodologies - an ability to exploit the simplicity and intuition behind RC methods without losing the input traffic characterization provided by NC methods. In order to do so, we identify the key sources of pessimism in RC methods, introduce the model and methods to characterize the traffic patterns. We then formulate a “Branch and Prune” algorithm which leverages these input characteristics in-order to eliminate packets which cannot arrive at run-time owing to task constraints and thus derive tighter bounds than the existing methods. We also propose a more general approach called “Branch, Prune and Collapse” which through a controllable parameter, offers the designer a trade-off between the tightness of the bound and computational complexity. By performing the simulations, we validate and verify the performance of our algorithm in comparison with the RC-based approach presented by [Ferrandiz et al. 2009] and observe that our approach dominates this method by yielding at least as tight as and in most cases tighter WCTT estimates than the related work. We investigate the influence of the trade-off parameter on the derived bounds.

3. SYSTEM MODEL

3.1. Platform and Application Model

Platform model. As previously noted, without loss of generality we drive our discussion in the context of tile-based platforms from Tilera. As seen in Figure 2, the tile-based architecture uses a 2D-mesh network to interconnect the processors and serves as the transport channel for off-chip memory access, I/O, interrupts, and other communication.

As illustrated in Figure 1, each tile comprises a general purpose processing engine (core), a cross-bar switch and a private cache. The platform is thus structured as a grid of $m \times m$ tiles, where "m" is the dimension of the square grid and r off-chip subsystems (e.g. memory controllers). The off-chip subsystems are connected to some of the tiles on the periphery of the grid. Inter-tile communication is achieved by routing packets via the embedded switches. Note that the terms router and switch are used interchangeably in the rest of the paper.

Generally, the switch that is embedded in each tile is part of several networks. Independent networks are typically used to handle different types of traffic to minimize the interference and maximize the performance. For example, the TILEPro$^{\text{TM}}$and TILE64$^{\text{TM}}$family of chips employ distinct networks to transmit traffic related to memory, caches, I/O and inter-tile communication between application. Since a packet can travel (and interfere with other packets) only over one of the networks, the analysis of the WCTT of a given packet can be carried out by considering each network individually. Hence, the analysis presented in this paper considers only the relevant inter-tile communication network.

The entire platform can thus be modeled by a directed graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where

— $\mathcal{N} = \{\mathrm{n}_1, \mathrm{n}_2, \ldots, \mathrm{n}_{2m^2+\mathrm{r}}\}$ is the set of $2m^2 + \mathrm{r}$ nodes comprising m^2 switches, m^2 cores and the r off-chip subsystems (caches are considered part of the processing cores) and
— $\mathcal{L}$ is the set of directed (physical) links that interconnect the switches to the cores, to other switches or to the off-chip subsystems.

For a given link $l \in \mathcal{L}$, we denote by $\mathrm{lsrc}(l)$ and $\mathrm{ldest}(l)$ the source and destination node of the directed link, respectively. A bi-directional link is modeled by using two links in opposite directions and all the links have the same capacity denoted by C. We assume that the links support full-duplex transmission with the interpretation that request and response packets can be simultaneously sent across a tile and will not contend amongst each other for the link. Our model is applicable to any generic platform which can be modeled as a graph and hence is not restricted to the Tile64 platform.

Application Model. As a first step, we assume that there is a 1:1 mapping between applications (called tasks hereafter) and cores; each task τ_i is non-preemptive, statically assigned to a dedicated core and does not migrate during its execution. We also assume that the cores do not support hyperthreading. The assumption of a single task is made to focus on the network latency delays, while efficiently abstracting away the problems of on-core interferences and dealing with the processor scheduling policies.

3.2. Switching and Routing Mechanism

Data is transmitted over the network, embedded in "packets". A packet comprises a header containing the destination address and a payload, which contains the actual data to be transmitted. In our model, packets are switched using the wormhole switching technique within which every packet sent over the network is split into smaller irreducible units called flits (FLow control digITS). The first flit of each packet is called the header flit: it stores the destination address and arbitrates for a given output port at a switch. Specifically, when a packet is granted access to an output port, it locks down that output port until its last flit has successfully traversed the switch. Since the subsequent (data) flits do not store any information about the destination, they always follow the same path as the header flit. When the output port is unavailable,

the flits remain buffered in finite (and typically small) sized buffers in the router, until the output port is freed. In order to ensure fairness in the arbitration, we assume that the switches implement round-robin arbitration as in [Tilera 2011]. We denote by d_{sw}, the time needed for arbitration and subsequently grant access to the output port to one of the pending packet. The value of d_{sw} is typically less than 25 μs for the Tilera platform [Tilera 2011].

We also consider that packets are routed statically using the XY (or YX dimension) routing algorithm. In XY routing, packets always travel in the X direction first, then the Y-direction. The XY routing algorithm is known to be deadlock and livelock free [Hu and Marculescu 2003] and is employed by many-core architectures like the Tile64 [Tilera 2011]. However, in general, our model can adapt to any static routing algorithm as long as it is *deadlock free*. While adaptive routing schemes are more efficient than the static ones, they are non-deterministic and hence are not considered here. A physical link that connects two routers (also referred to as channel) may be split into several *virtual* channels to allow multiple packets to pass through in parallel. In the analysis that follows, as a first step, we assume that every physical link implements only a single virtual channel hence allowing only a single packet at every input port of a router.

3.3. Communication and traffic modeling

3.3.1. The basic flow model. The network traffic between two tasks or between a task and an off-chip subsystem is modeled by a *flow*. Each flow f is characterized by an origin and a destination node, denoted by $\mathrm{fsrc}(f)$ and $\mathrm{fdest}(f)$ (respectively) and a maximum packet size denoted by $\mathrm{maxpsize}(f)$. In order to reach its destination, every packet of a flow f is routed throughout the network over a pre-defined static *path* defined by an ordered list of *links* and denoted by $\mathrm{path}(f)$. The number of hops traversed by the packets of f along this path is given by $\mathrm{nhops}(f)$. Also, we denote by $\mathrm{first}(f)$ the first link of $\mathrm{path}(f)$ and we use the notations $\mathrm{prev}(f,l)$ and $\mathrm{next}(f,l)$ to refer to the links directly *before* and *after* the link l in $\mathrm{path}(f)$, respectively. Finally, $\mathcal{F}$ denotes the set of all the flows in the system.

3.3.2. Our extended model. We augment the simple model given above by distinguishing between two types of packet-*release* profiles, namely *regulated* (Reg) and *unregulated* (UnReg) flows. A *regulated* flow models a sporadic communication between two nodes, with the interpretation that a packet of a regulated flow f can be released *at least* a certain duration *after the receipt of the previous packet of the same flow has been acknowledged by the receiver at the destination node*. This minimum time duration is referred to as the *minimum non-sending time* of the regulated flow f and is denoted by $\mathrm{MinNonSend}(f)$. In practice, $\mathrm{MinNonSend}(f)$ represents the application-specific delay (on the core) before another packet can be generated: it may be an explicitly defined waiting phase or time spent for processing. A stream of video frames which must be transferred to an off-chip graphic controller is an example of a regulated flow. An *unregulated* flow in contrast, models an aperiodic communication between two nodes: the source node can release a packet at any instant in time *after the receipt of its previous packet has been acknowledged*, i.e., for all unregulated flows f, $\mathrm{MinNonSend}(f)$ is null. Data transfers between a task and the system memory at arbitrary times due to random cache misses serves as an example of unregulated flows. It is important to reiterate that our model inherently assumes "blocking" communication: the next packet of a given flow can be generated only after the receipt of the acknowledgement of the previous packet.

4. INPUT TRAFFIC CHARACTERIZATION FUNCTIONS

In this section, we introduce two functions associated with each flow, namely the *minimum inter-release time* function and the *maximum packet release* function.

4.1. The minimum Inter-Release Time Function

DEFINITION 1. *The Minimum Inter-Release Time function* $\mathrm{MinInterRel}(f)$ *of a flow* f *is the minimum time gap between two consecutive packets released by* f.

Specifically, if p_1 and p_2 are two consecutive packets generated by flow f, then $\mathrm{MinInterRel}(f)$ is the sum of (i) the minimum time needed to deliver *and acknowledge* p_1 — sometimes referred to as the round-trip time of p_1 — and (ii) the application-specific minimum delay that must elapse before the release of p_2, i.e., $\mathrm{MinNonSend}(f)$. We then compute $\mathrm{MinInterRel}(f)$ as:

$$\mathrm{MinInterRel}(f) = \mathrm{MinDest}(f) + \mathrm{MinDest}(ack) + \mathrm{MinNonSend}(f) \tag{1}$$

where $\mathrm{MinDest}(f)$ is the minimum time taken by a packet of f to travel from its source $\mathrm{fsrc}(f)$ to its destination $\mathrm{fdest}(f)$. Note that $\mathrm{MinInterRel}(f)$ differs from $\mathrm{MinNonSend}(f)$ as it also includes the minimum time needed to acknowledge a packet of f. As a direct consequence of the need for acknowledging a packet, it can be easily shown that no two packets of the same flow f can reach a given router separated by a time gap of less than $\mathrm{MinInterRel}(f)$ time units. Note that these parameters are computed in isolation (i.e. without any contention from the other flows). Since the objective is to determine the WCTT of a given packet (say p), we must be able to capture the worst-case scenario in which the blocking flows can cause maximum interference to any packet of the analyzed flow. We must therefore have a parameter which represents a *lower-bound* on the inter-release time of all these blocking flows and so a lower-bound on $\mathrm{MinDest}(f)$ for all flows $f \in \mathcal{F}$. We shall use the following result.

DEFINITION 2. *For the wormhole routing technique, the minimum time-to-destination* $\mathrm{MinDest}(f)$*, for any given flow* $f \in \mathcal{F}$*, is given by*

$$\mathrm{MinDest}(f) = \mathrm{nhops}(f) \times (d_{\mathrm{sw}} + d_{\mathrm{across}}) + \frac{\mathrm{minpsize}(f)}{C} \tag{2}$$

where d_{across} *is the time for a flit to be read from an input buffer, traverse the crossbar (the switch) and reach the storage at the input of a neighboring switch.*

The above equation can be interpreted as follows. The term $\mathrm{nhops}(f)$ denotes the number of hops that the first (header) flit of the packet of f traverses while travelling from its source to destination. While traversing the network, the first flit locks down all the output ports on its path and at each intermediate switch, it incurs an arbitration delay of d_{sw} and a time of d_{across} to traverse the crossbar. In our model, d_{across} also accounts for the maximum time it takes to transfer flow-control tokens between the routers. Once this first flit reaches the destination, all the traversed output ports from its source to its destination have been locked down and the entire packet of size $\mathrm{minpsize}(f)$ can travel over the network of capacity C, which requires $\mathrm{minpsize}(f)/C$ time units. Hereafter, Equation (2) will be used as the value of $\mathrm{MinDest}(f)$.

4.2. The Maximum Packet-Release Function

DEFINITION 3. *The Maximum Packet Release Function* $\mathrm{MaxPcktRel}(f, t)$ *of a flow* f *provides an upper-bound on the number of packets that* f *can generate in a time interval of length* t.

This function is computed considering that the task (initiating the flow) is run in isolation, i.e., without any contention from other packets on the network and hence can be determined at design time. Methods to compute an upper bound on the number of requests issued by a task in a given time interval have been proposed by [Dasari et al. 2011], [Schliecker et al. 2010] and [Pellizzoni et al. 2010]. For *regulated* flows, the maximum packet release function can be expressed based on the minimum inter-release time of the given flow as in Equation (3), since their minimum non-sending

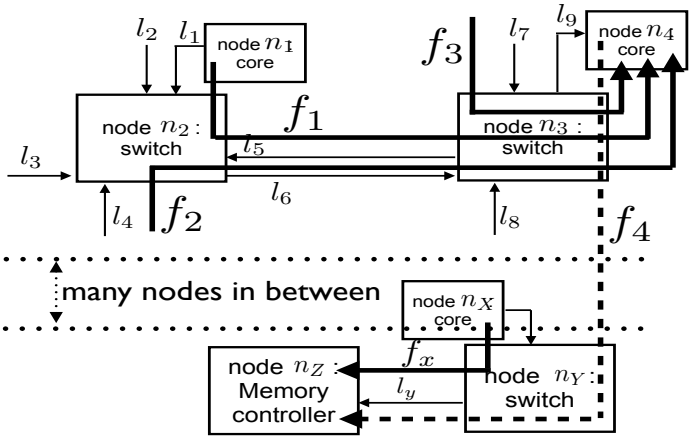


Fig. 3. Example to illustrate network and task-level pessimism

time is clearly defined and integrated into their minimum inter-release time.

$$\mathrm{MaxPcktRel}(f,t) = \left\lceil \frac{t}{\mathrm{MinInterRel}(f)} \right\rceil \tag{3}$$

However, for *unregulated* flows, computing $\mathrm{MaxPcktRel}()$ with the same approach may lead to an over-estimated number of packets, especially for large values of t. To overcome this pessimism in the computation, we can apply the method proposed in [Dasari et al. 2011]. Although the exact time-instants at which unregulated flows generate packets are not known, these methods calculates this parameter by instrumenting the task code at different sampling points when the task executes in isolation and uses this information to derive an upper bound on the maximum number of packets it can generate in any time interval of duration t. Note that the $\mathrm{MaxPcktRel}(f,t)$ function roughly corresponds to the arrival curve abstraction used in network calculus theory.

5. CONCEPTUAL DESCRIPTION OF EXISTING RC BASED METHODS

To understand the concepts behind the recursive calculus based method, we present an algorithm to compute an upper bound on the traversal time of a packet of flow f from its source to the destination node. This will represent the approach proposed by [Ferrandiz et al. 2009] and is conceptually similar to that of [Rahmati et al. 2009].

Algorithm 1: $d(f,l)$

input : a flow f, a link l
output: WCTT of f, starting from link l, to the destination.
// there cannot be any contention on the first link.
if $l = \mathrm{first}(f)$ **then return** $d(f, \mathrm{next}(f,l))$;
/* Header flit reaches end of path and the entire packet transits */
if $l = \mathrm{null}$ **then return** $\frac{\mathrm{maxpsize}(f)}{C}$;
/* Determine the set of links excluding $\mathrm{prev}(f,l)$ and whose destination node is $\mathrm{lsrc}(l)$ */
$\mathrm{BL} \leftarrow \{l_{\mathrm{in}} \in \mathcal{L} \mid l_{\mathrm{in}} \neq \mathrm{prev}(f,l) \wedge \mathrm{ldest}(l_{\mathrm{in}}) = \mathrm{lsrc}(l)\}$;
foreach $l_{\mathrm{in}} \in \mathrm{BL}$ **do**
  /* Determine the set of flows f_{in} that use link l_{in} and have l as the next link */
  $U_{l_{\mathrm{in}}} \leftarrow \{f_{\mathrm{in}} \in F \mid l_{\mathrm{in}} \in \mathrm{path}(f_{\mathrm{in}}) \wedge \mathrm{next}(f_{\mathrm{in}}, l_{\mathrm{in}}) = l\}$;
end
$\mathrm{delay} \leftarrow \sum_{l_{\mathrm{in}} \in \mathrm{BL}} \max_{f_{\mathrm{in}} \in U_{l_{\mathrm{in}}}} \{d_{\mathrm{sw}} + d_{\mathrm{across}} + d(f_{\mathrm{in}}, \mathrm{next}(f_{\mathrm{in}}, l))\}$;
return $\mathrm{delay} + d_{\mathrm{sw}} + d_{\mathrm{across}} + d(f, \mathrm{next}(f,l))$;

Let us consider the part of Figure 3 above the horizontal dotted lines. There are four nodes: two cores n_1 and n_4, two corresponding switches n_2 and n_3, and three flows: f_1, f_2 and f_3. All the flows terminate at the core n_4. Flow f_1 originates in n_1 and the source nodes of f_2 and f_3 are not specified in this example.

Let us compute the WCTT of flow f_1 by invoking the function $d(f_1, l_1)$ (Algorithm 1). Since link l_1 is the first link in the path of flow f_1, it could be blocked only if other flows generated by its source (core n_1) have to transit first. This particular case has been handled in earlier methods but in contrast to their approach, we assume that the cores stall while waiting for a given packet transmission to be completed before initiating a new transmission. Therefore, under this assumption, a flow f issued from one core can never be blocked by another flow issued from the same core and the first flit of the packet is directly transferred to the (top) input port of the switch n_2. Thus, the algorithm directly calls the function $d(f_1, l_6)$ at line 1. At this stage, the flow f_1 traverses via link l_1 and next passes through l_6 via node n_2. At line 3, the algorithm computes the set of links, BL, connected to the other input ports of n_2 (i.e., the links excluding l_1), Here, $\mathrm{BL} = \{l_2, l_3, l_4, l_5\}$. Then, for each of the links $l_{\mathrm{in}} \in \mathrm{BL}$, the algorithm determines the set $U_{l_{\mathrm{in}}}$ of blocking flows which passes consecutively through links l_{in} and l_6. Here $U_{l_4} = \{f_2\}$ and the other sets do not have flows matching the criterion stated above and are therefore empty. Note that exactly one blocking flow of each set $U_{l_{\mathrm{in}}}$ may block f_1 since the switch arbitration rule is assumed to be round-robin. Therefore, to maximize the delay, for each link l_{in} in BL, the algorithm explores all the flows f_{in} in $U_{l_{\mathrm{in}}}$ by recursively invoking $d(f_{\mathrm{in}}, next(f_{\mathrm{in}}, l))$ and then chooses the flow which maximizes the delay in line 6. It then computes the cumulative delay by summing up the maximum delays obtained for each $l_{\mathrm{in}} \in \mathrm{BL}$. After the blocking flows are allowed to progress at the current link, the flow being analyzed, f_1 can progress to its next link. At line 7, the algorithm returns the cumulative delay computed in line 6, plus the time for the flow f_1 to traverse through n_2 (i.e., d_{sw}), plus the delay suffered by f_1 in the next hop, i.e., $d(f_1, next(f_1, l_6)) = d(f_1, l_9)$. Notice that at line 2, if $l = \mathrm{null}$, then it implies that the flow f has reached its destination. In this case, the packet of f is fully transmitted, which requires, in the worst-case, $\frac{\mathrm{maxpsize}(f)}{C}$ time units.

6. PROPOSED METHOD FOR TIGHTER WCTT

In order to compute a tighter WCTT, we first explore the sources of pessimism in the previous approach and then describe the methods to deal with it.

6.1. Sources of Pessimism

Although the computation presented in the previous section is correct and terminates within a reasonable computation time (as shown by [Ferrandiz et al. 2009]), we identified two main sources of pessimism. In order to highlight this pessimism, we constructed a computation tree as shown in Figure 4 based on Algorithm 1. In this tree, each recursive call to the function $d(f, l)$, with $l \neq \mathrm{null}$, is a (non-leaf) node of the tree and each call to $d(f, \mathrm{null})$ is a leaf node. Algorithm 1 traverses this computation tree in a *pre-ordered depth-first* manner: first the root node is visited and then each of the children are visited, from the left to the right.

As seen in Figure 4, the order in which the leaf nodes of the computational tree are reached reflects the following scenario. The flow f_1 is delayed because f_2 goes first through l_6 (step ①). f_2 is then blocked by f_3 at node n_3. Once f_3 has reached the core n_4, its whole packet is transferred to n_4, hence adding $\mathrm{maxpsize}(f_3)/C$ to the delay (step ②, the first "leaf"). Then f_2 flows and reaches the core n_4 (step ③), followed by f_1 which passes through n_2 but gets blocked by another flow of f_3 in n_3. This second flow of f_3 passes first (step ④) and finally f_1 can progress to its destination (step ⑤).

As a conclusion, the scenario considered by the computation of $d(f_1, l_1)$ assumes that f_3 blocks the flow f_1 twice before it finally reaches the core n_4. These multiple blockings may not be possible for several reasons and can lead to an overestimation of the WCTT.

In the next subsection we explore these reasons which we refer to as the "sources of pessimism" and propose methods to overcome them.

6.2. Network-level Pessimism

The basic premise behind earlier approaches was the assumption that every flow injects traffic continuously into the network, thereby assuming that for all flows, the packets do not expect any response and have no temporal constraints on their generation. In practice, the application initiating a flow may dictate that two consecutive packets cannot be generated separated by less than a minimum time gap (given by the $\mathrm{MinInterRel}()$ functions). As seen in the previous section, this minimum time gap cannot be null as it includes the round-trip communication delay incurred due to the underlying network. In a large setup, on ignoring these "minimum round-trip time" constraints, the delay incurred as a result of these non-feasible contentions can cascade and increase in magnitude as the flow path unrolls, ultimately leading to highly pessimistic WCTT estimates.

In the example of Figure 3, it is seen that flow f_3 blocks f_1 twice: once indirectly by blocking f_2 and once directly in step ④. However, because of its round-trip time constraint it may not be possible for flow f_3 to release a second packet by the time flow f_1 transits through node n_3. Thus by taking into account this constraint, the analysis of the delay incurred by f_1 can be less pessimistic.

In order to tackle this source of pessimism, we introduce the notion of "current time" during the computation. The current time is initialized to 0 at the beginning of the analysis and it is then increased by $d_{\mathrm{sw}}+d_{\mathrm{across}}$ every time a flow traverses a router and by $\mathrm{minpsize}(f_x)$ whenever a flow f_x reaches its destination. During the computation, whenever a flow f_x traverses a router n_k, the current time (denoted by t) is recorded and used as a time-stamp for this traversal, i.e., a time-stamp is attached to the pair $< f_x, n_k >$. Then, in the time interval $[t, t + \mathrm{MinInterRel}(f_x)]$, our proposed analysis will not recognize flow f_x as a potentially blocking flow in router n_k as it is not possible for f_x to have another packet at an input port of n_k in that interval of time.

6.3. Task-level Pessimism

Intuitively, it may happen that many occurrences of a same flow f_x have to be considered by Algorithm 1 and none of them violates its $\mathrm{MinInterRel}(f_x)$ constraint, i.e., at every router on f_x's path, every occurrence of f_x is separated in time from the previous one by at least $\mathrm{MinInterRel}(f_x)$ time units.

A manifestation of this situation can be seen in the example of Figure 3 (consider the portion below the dotted line). Let us assume that the destination of the (dashed) flow f_4 is the memory controller denoted by the node n_Z, and that n_Z is distantly located from n_4 (in terms of number of hops). In addition, consider a flow f_x from the core n_X to n_Z. When computing the WCTT of f_4, chances are high that Algorithm 1 invokes the function $d(f_x, l_y)$ a significant number of times since it blocks all the flows directed to

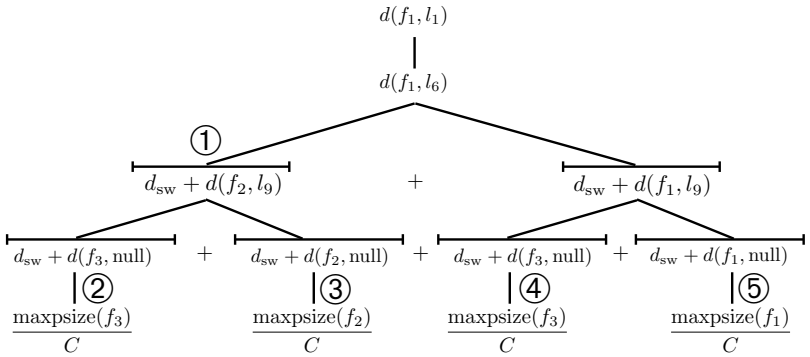


Fig. 4. Computation tree of $d(f_1, l_1)$.

n_Z. As stated above, in such a case the flow f_x may never violate the constraint imposed by $\mathrm{MinInterRel}(f_x)$, as the distance between n_X and n_Z is short, but at a given current time t during the analysis it may be the case that the task which initiates f_x is not able to generate that many packets in the time interval $[0, t]$. As a result, Algorithm 1 may return overly pessimistic WCTT estimates as it does not take into account the packet-release profile specific to each task.

The paper proposes to tackle this source of pessimism by extending the solution for tackling the network-level pessimism. Instead of recording and maintaining a time-stamp only for the *last* occurrence of every flow f_x in every router n_k, we propose to save a time-stamp for *every* occurrence of all the flows in every router that they traverse. That is, for each pair of flow-router $< f_x, n_r >$, we maintain the number of occurences and the corresponding list of time-stamps reflecting all the time-instants at which f_x has traversed n_k during the computation. Let us assume t_0 is the time-stamp of the first occurrence of f_x in n_k. Then, at any current time t during the analysis, the flow f_x is deemed infeasible (and hence cannot potentially block the analyzed flow another time) if the number of recorded occurences of f_x in router n_k exceeds $\mathrm{MaxPcktRel}(f_x, t - t_0)$. As described earlier, this upper bound on the number of packets is given by the function $\mathrm{MaxPcktRel}(f_x, t - t_0)$ defined in Section 3.3.2.

7. THE BRANCH AND PRUNE ALGORITHM

This section introduces our "Branch and Prune" (BP) algorithm for calculating the WCTT of a packet released by a flow. While the basic principle of Algorithm 1, which consists of recursively tracking the progression of all flows throughout the network remains the same, our method differs from this algorithm in two aspects:

(1) It considers the extended flow model presented in Section 3.3.2, in which the input traffic of the flows are characterized with specific functions and
(2) It incorporates the ideas described in Section 6 to reduce the task- and network-level pessimism.

7.1. Overview of the Branch and Prune Algorithm

The basic principles of flow progression remain the same as in Algorithm 1. At each link l in the path of the analyzed flow f, we first determine the set of all flows that can potentially block f by accessing l first. Then, we enumerate all possible interfering scenarios for that link and we analyze each of them recursively. An interfering scenario is defined here as a flow sequence, i.e., an order of passage of the blocking flows over the considered link.

One of the main differences with Algorithm 1 is that we first branch-out, thereby enumerating all possible blocking flow sequences, then we validate if the flows in the sequence can arrive, given their task constraints and thereby prune the infeasible flows in each sequence. We compute and record the traversal times of these pruned sequences. Since the tests for constraint compliance are applied early-on in the computation, the resulting search space is greatly reduced. This is especially true for loaded networks where in the impact of indirect contention of certain flows can cause the search space to grow exponentially. It can be seen that pruning an infeasible flow is equivalent to pruning the entire subtree of flows which would have blocked it (which would have to be otherwise explored by the algorithm increasing the search space).

7.2. Concepts behind the algorithm

The main steps of our approach are described here with a simple example, illustrated in Figure 5. Let us assume that f is the analyzed flow which traverses on its kth hop through the router n_x along its path to the destination. This implies that interfering flows of an earlier hop may have been already analyzed for their delay in node n_x.

7.2.1. Blocking Links and Flows. At every router in the path of f, we first determine the set of blocking links. In the given example, the set BL of blocking links at router n_x is

given by $\mathrm{BL} = \{l_p, l_q\}$. Then, for each blocking link l_{in}, we compute the associated set $U_{l_{\mathrm{in}}}$ of blocking flows that can potentially block f in this router n_x. Here, $U_{l_p} = \{f_a, f_b\}$ and $U_{l_q} = \{f_c, f_d\}$.

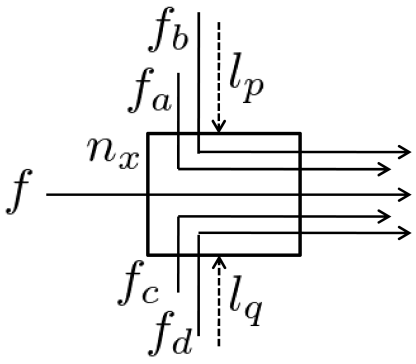


Fig. 5. A simple example.

At this stage, we still assume that all these blocking flows have a packet to transmit and are allowed to transmit it *before* the flow f progresses.

7.2.2. List of Interfering Scenarios. During the computation, the progress of all flows is ensured by the round-robin arbitration policy applied at each intermediate router, which implies that at most one packet from each of the blocking flows can block the analyzed flow f. At this phase, we are interested in finding the sequence of flows progression over the network that will delay f for as long as possible. To tighten the WCTT estimate by overcoming the limitations (sources of pessimism) presented earlier, at each router in the path of the analyzed flow we first enumerate all the possible flow sequences (called "interfering scenarios" hereafter) that might block it. Specifically, we denote by LIS the list of all possible "Local Interfering Scenarios" at the current router, assuming that the currently analyzed flow f is the last one allowed to progress. That is, at router n_x for example, LIS contains $\{f_a, f_c, f\}$, $\{f_a, f_d, f\}$, $\{f_b, f_c, f\}$, $\{f_b, f_d, f\}$, $\{f_c, f_a, f\}$, $\{f_c, f_b, f\}$, $\{f_d, f_a, f\}$, $\{f_d, f_b, f\}$, $\{f_a, f\}$, $\{f_b, f\}$, $\{f_c, f\}$, $\{f_d, f\}$, and $\{f\}$.

Note: Upon explicitly pruning all the computed flow sequences and removing the duplicates which appear as a consequence of the pruning mechanism, it is necessary to investigate all of them, so as to eliminate any possible timing anomalies and ensure the safety of the algorithm. The necessity comes from the fact that any of the scenarios at a given router cannot be implicitly discarded, as the local maximum at a given router may not translate into the global maximum. For example, in the above LIS, we cannot ignore the scenarios $\{f_a, f\}$ – intuitively we may conclude that among scenarios $S1 = \{f_c, f_a, f\}$ and $S2 = \{f_a, f\}$, the scenario S1 is more likely to contribute to the WCTT, but it may not be so. In fact, $S2 = \{f_a, f\}$ may in the future progression, allow for flows contributing to a large interference to be included in the final WCTT while scenario S1 may prohibit it thus leading to a timing anomaly and also leading to an unsafe WCTT.

We traverse this LIS and investigate each interfering scenario individually. When considering $\{f_a, f_c, f\}$ for example, we first compute (recursively) the WCTT of f_a, then the WCTT of f_c and finally we allow f to progress to the next router on its path. However, before computing the WCTT of f_a (and the same holds for f_c later), we reduce the pessimism of the computation by applying the two optimization mechanisms that determine whether f_a is "feasible". Being infeasible implies that it is impossible for a given flow to release a packet at the given time, considering that it either released a packet too close in time relative to its previous packet (thereby accounting for the network-level pessimism) or it has already exceeded the upper bound on the number of packets it could possibly generate from the beginning of the computation (task-level pessimism). If the flow f_a is declared *infeasible* then it is removed from the currently considered interfering scenario $\{f_a, f_c, f\}$ and the algorithm moves on with the next flow f_c of that scenario. As a side note, it can be observed that removing f_a from that scenario $\{f_a, f_c, f\}$ yields the scenario $\{f_c, f\}$ and thus, the equivalent scenario $\{f_c, f\}$ listed in LIS will not have to be investigated again later on. Thereby, removing flows from a scenario is equivalent to pruning the resulting scenario from LIS, which considerably improves the time-complexity of this technique (as well as the accuracy of the

WCTT estimates) as discarding a single flow from a scenario automatically cuts off a whole subtree from the computation tree.

7.2.3. Need for ordering. Since our approach considers the input flow characteristics, the order in which flows progress cannot be ignored as it can lead to different results. We illustrate this with an example given in Figure 6. Consider two possible scenarios: $S_1 = \{f_c, f_d, f_b, f_d, f_c, f_a, f_c, f_d, f\}$ and $S_2 = \{f_c, f_d, f_b, f_c, f_d, f_a, f_c, f_d, f\}$. These two scenarios only differ in the order in which f_c and f_d block f_a. However, notice that in S_1 the first and the second appearance of f_d are distanced only by f_b, while the second and the third appearance of f_c are distanced only by f_a. Conversely, in S_2, any two appearances of the same flow are distanced by at least two other flows. Depending on flow characteristics, in some cases the entire S_2 might be feasible, while S_1 would require the pruning of some appearances of f_c and/or f_d. Thus, considering only S_1 in the analysis may result in unsafe worst-case estimates, and in order to capture the worst-case it is necessary to investigate all possible flow orderings at every traversed router.

Indeed, at a given router n_x along f's path, the list of interfering scenarios can be computed as explained above but identifying which blocking flows are *infeasible* within each of these scenarios requires the knowledge of which flows have already progressed (and in which order) in the previously traversed routers, before f reaches n_x. Without this knowledge, our pruning mechanisms would not be able to determine whether a flow listed in an interfering scenario is feasible or not. This leads to the concept of "context" which is key in our algorithm.

7.2.4. Notion of a "context". Formally, a context is a snapshot of all the information characterizing a unique sequence of flow progressions throughout the network before a given flow f reaches a given router n_x, including (1) the order in which the flows have progressed over the network so far, (2) all the past occurrences and the associated timestamps of all these flows in this flow sequence and (3) the delay incurred by f before it reaches n_x. At a given router n_x, for a given flow f and context ctx, which (informally speaking) reflects the history of what happened in the network before that flow f reached n_x, we explained above that every feasible blocking flow of every interfering scenario of LIS in n_x will be allowed to progress from n_x to its destination *before* the analyzed flow f. Therefore, it can be seen that the progression of each of these blocking flows towards their destination may in-turn generate a multitude of new contexts (more exactly, the progression of each blocking flow will make the current context ctx evolve in an unique way). Subsequently, all these new contexts derived from ctx are investigated when f eventually progresses to the next router on its path. At the end, the WCTT of f will be found by looking at all the flow sequences (i.e., the contexts) in which f finally reaches its destination.

7.3. Detailed Explanation of the Algorithm GetContexts()

Initially, the algorithm is invoked by Algorithm 3 with the following inputs: the flow f to be analyzed, the first link l on its path and an initial "context". The initial context contains an empty flow sequence, a delay of zero, and past occurrences set to null. On arriving at a router, the algorithm (line 10) computes the set of blocking links BL (as explained earlier) and the corresponding blocking flows (lines 11-13) incident on each of the links in BL. Based on this information, the set LIS of *Local Interfering Scenarios* is computed (line 14) as follows: LIS contains all permutations of flow sequences in which (i) there is exactly zero or one flow from each set $U_{l_{\mathrm{in}}}$ and (ii) the analyzed flow f is appended to each of these permutations.

Once the set LIS is computed, the algorithm investigates each of them. For each scenario in LIS, (line 16), e.g, $\{f_a, f_c, f\}$, the list SCList will ultimately contain all the generated contexts arising from the execution of this scenario. The investigation starts with the first flow, here f_a (line 18), and considers every current context curCtx (line 19) that results in f reaching the link l. Remember that, (because our extended model considers flows with some timing constraints on their packet generation) the interfering

Algorithm 2: getContexts(Flow f, Link l, Context curCtx)

input : a flow f, a link l, a context curCtx
output: A set of contexts, each constituting a scenario string and its delay

if $l = \text{null}$ **then**
  curCtx . delay += $\frac{\text{maxpsize}(f)}{C}$;
  curCtx.scenario.Append(f) ;
  return curCtx;
end
if $l = \text{first}(f)$ **then**
  curCtx . delay += $d_{\text{sw}} + d_{\text{across}}$;
  return getContexts($f, next(f, l)$, curCtx) ;
end
BL $\leftarrow \{l_{\text{in}} \in \mathcal{L} \mid l_{\text{in}} \neq \text{prev}(f, l) \wedge \text{ldest}(l_{\text{in}}) = \text{lsrc}(l)\}$;
foreach $l_{\text{in}} \in$ BL **do**
  // Find the set of blocking flows on link l_{in}
  $U_{l\text{in}} \leftarrow \{f_{\text{in}} \in F \mid next(f_{\text{in}}, l_{\text{in}}) = l\}$;
end
LIS $\leftarrow$ Set of local interfering scenarios based on $U_{l_{\text{in}}}$;
GCList $\leftarrow \{\emptyset\}$;
foreach *scenario* $S_i \in$ LIS **do**
  SCList $\leftarrow$ {curCtx};
  foreach *flow* $f_j \in S_i$ **do**
    foreach *context* $\text{ctx}_k \in$ SCList **do**
      SCList .*pop*(ctx_k) ;
      if isMITRCompliant(f_j, ctx_k . LogTbl, ctx_k .*delay*) **then**
        if isMPRFCompliant(f_j, ctx_k . LogTbl, ctx_k .*delay*) **then**
          ctx_k . delay += $d_{\text{sw}} + d_{\text{across}}$;
          $\text{FCList}_k \leftarrow$ getContexts($f_j, next(f_j, l)$, ctx_k);
        end
      end
    end
    SCList $\leftarrow \cup_{\forall k}$ FCList_k;
  end
  GCList $\leftarrow$ GCList $\cup$ SCList ;
end
return GCList;

Algorithm 3: getMaxDelay(Flow f, Link l)

input : a flow f, a link l
output: WCTT of flow f, starting from link l, to the destination.

StudiedFlow $\leftarrow f$, ctx.scenario $\leftarrow$ " ", ctx . delay $\leftarrow 0$;
ctxSet $\leftarrow$ getContexts(f, l, ctx) ;
maxdelay $\leftarrow \max_{\forall\ \text{ctx} \in \text{ctxSet}}$ ctx . delay ;
return maxdelay ;

Algorithm 4: isMPRFCompliant(fid,flowLogTbl,curTime)

numGen $\leftarrow$ flowLogTbl [fid]. numOccurences ;
firstArrival $\leftarrow$ timeStampArray[1] ;
timeDuration $\leftarrow$ curTime $-$ firstArrival ;
numMax $\leftarrow$ MaxPcktRel(fid, timeDuration);
if (numMax $<$ numGen) **then return** FALSE ;
return TRUE;

scenarios that can occur for a flow f in a router n_x depend on the order in which the previous flows progressed through the network before f reaches n_x. Hence, whenever a flow f reaches a router n_x, all possible contexts have to be investigated in order to determine all the future scenarios that could arise at router n_x.

Algorithm 5: UpdateLogTbl(f,flowLogTbl,curTime)

```
   // For the entry of flow f in flowLogtbl
1  Increment numOccurences ;
2  Append curTime to the timeStampArray ;
3  Reset NextfeasibleArrival to curTime + MinInterRel(f) ;
```

Algorithm 6: isMITRCompliant(fid,flowLogTbl,curTime)

```
if (FirstOccurence(fid, flowLogTbl))||(curTime ≥ fid.NextfeasibleArrival) then
    UpdateLogTbl( fid, flowLgTbl, curTime) ;
    return TRUE ;
end
return FALSE ;
```

First, lines 21 and 22 check whether f_a can legally block the analyzed flow f considering the time of its last arrival in the considered context (here, curCtx). If f_a can arrive and passes the round trip time test (line 21) as described in Algorithm 6, the task characteristics of the flow are checked. Specifically, line 22 checks if the task originating f_a can indeed generate that many packets in the time specified by calling the function defined in Algorithm 4. If both checks of lines 21 and 22 succeed, then f_a is allowed to progress in line 24, after updating the current context delay parameter (line 23). Ultimately, the passage of f_a returns a set of contexts (line 24), when flow f_a reaches its destination (which is reflected when its last link is null (lines 1-5)).

All these returned contexts end with flow f_a reaching its destination and each one corresponds to a different scenario in the subsequent routers along the path of f_a. All these resulting contexts are added to the scenario list SCList (line 28) *as they must all be considered (line 17) while analyzing the next flow f_c of the current scenario*. When all the flows of the currently considered interfering scenario have been considered, all the contexts resulting from this scenario are added to the global list GCList (line 30). That list is finally returned (line 32), as it contains all the possible flow sequence progressions in the routers traversed by f after progressing through the link l, starting with the context curCtx. The list GCList that is ultimately returned (to Algorithm 3) at the end of the analysis, contains all possible scenarios in which f can reach its destination, with the corresponding delays. Finally, Algorithm 3 selects the scenario with the highest delay to return an upper-bound on the traversal time of the analyzed flow f.

8. A MORE EFFICIENT ALGORITHM: BRANCH, PRUNE AND COLLAPSE

8.1. Description

The recursive calculus based method proposed by [Ferrandiz et al. 2009] scales well at the cost of providing a very pessimistic WCTT. In contrast, the proposed branch and prune (BP) algorithm returns a very tight WCTT at the cost of a high time complexity and memory usage. The reason for the improved computational efficiency in the method by [Ferrandiz et al. 2009] is that it does not carry any history of the previous flows and only retains only the maximum delay incurred at each router. From the two extreme approaches it may be inferred that a hybrid solution exists which can drop some history of the contexts (only) periodically while retaining the maximum delay seen so far as the analysis progresses. Such an approach is explored in this section.

As seen earlier, the identification of infeasible scenarios in BP was possible due to the explicit book-keeping of contexts of *all* investigated scenarios. The "Branch, Prune and Collapse" (BPC) algorithm presented (conceptually) in this section is motivated by the observation that, while the complexity of the BP algorithm is indeed exponential, for most flows, the number of scenarios to be considered is manageable (as seen in the experiments of Section 9). To handle the corner cases (in terms of time complexity) we propose as a trade-off, a more general BPC algorithm with a tunable parameter that

we term "Scenario Information Retention Limit" (SIRL). The SIRL acts as a threshold on the number of scenarios whose contexts are retained.

In the BP algorithm, all the investigated scenarios and their contexts are back propagated and the algorithm proceeds by combining them with the local scenarios after pruning it and then allowing the next feasible flow to progress. As a deviant version of this algorithm, which is hereafter referred to as BPC, *when the number of investigated scenarios reaches a pre-set limit of SIRL*, a dummy scenario with a unique dummy flow-id is created. The context of this scenario populates (i) the delay field to the maximum of the delays of the investigated scenarios and (ii) the other fields, relating to the history information i.e the past occurrences, timestamps, etc to NULL (or zero, as appropriate) – This marks the "collapse" phase of the BPC algorithm in which a set of investigated scenarios is "collapsed" into a new single dummy scenario with zero history information and a conservative delay estimate. As opposed to the branch and prune algorithm, only this dummy scenario is back propagated to the higher nodes and the algorithm gets back in to the branch and prune phase until the number of investigated scenarios again exceeds SIRL, thereby triggering another collapse phase.

The necessity to create a new dummy scenario. Note, that at an *intermediate* stage of analysis when the SIRL reaches its pre-set threshold, a single scenario that will *provably* lead to the WCTT cannot be detected. That is, during the collapse phase, from the set of the constituent collapsed scenarios, a specific single scenario (containing a tightly coupled local maximum delay, flow sequences and other history information) cannot be specifically carried forward. This is due to the fact that in the later analysis stages such a scenario with the local maximum might be subject to pruning because of its flow history and thereby not contribute to the global maximum delay. In order to prevent this, we drop the history information (thereby reducing the chances of further optimization due to the loss of history retained in the collapsed scenarios) while only retaining the local maximum delay in a totally new dummy scenario. To summarize, the BPC method thereby creates a dummy scenario which inherits the delay of the local maximum, but drops the history of the flows constituting that scenario (context).

8.2. An example to illustrate the BPC method

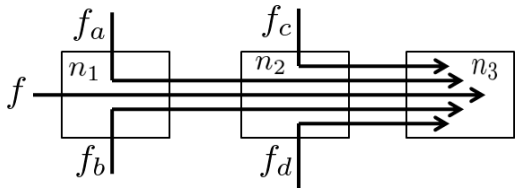


Fig. 6. Example for "Branch, Prune and Collapse"

We illustrate the working of the BPC method with the example of the flow-set presented in Figure 6 – let us analyze flow f which traverses through routers n_1 and n_2 to finally reach its destination n_3. As observed in Figure 6, flows f_c and f_d can potentially block flow f thrice: twice indirectly by blocking the passage of f_a and f_b at n_2 (f_a and f_b block f directly at node n_1), and finally directly at n_2 during f's passage. Thus, f_c and f_d are promising candidates for pruning (lines 21 and 22 of Algorithm 2). Additionally, let us assume that $\mathrm{SIRL} = 5$. At node n_1, BPC constructs the LIS as $\mathrm{LIS}(f, n_1) = \langle\{f_a, f_b, f\}, \{f_b, f_a, f\}, \{f_a, f\}, \{f_b, f\}, \{f\}\rangle$ and it starts exploring the first scenario $\{f_a, f_b, f\}$. At this time, the list SCList is reset to the current context (at line 17 of Algorithm 2). Note, that the list SCList will ultimately contain all generated contexts arising from the execution of this scenario $\{f_a, f_b, f\}$, before being appended to the global list of contexts GCList at line 30. Firstly, a recursive call is performed to node n_2 with f_a being the analyzed flow. This will result in a new LIS constructed at node n_2 as $\mathrm{LIS}(f_a, n_2) = \langle\{f_c, f_d, f_a\}, \{f_d, f_c, f_a\}, \{f_c, f_a\}, \{f_d, f_a\}, \{f_a\}\rangle$. Similarly, LIS is generated for the flow f_b at n_2, resulting with $\mathrm{LIS}(f_b, n_2) = \langle\{f_c, f_d, f_b\}, \{f_d, f_c, f_b\}, \{f_c, f_b\}, \{f_d, f_b\}, \{f_b\}\rangle$.

These scenarios are back-propagated to the node n_1, and should be combined before f progresses to n_2 itself. As both LIS sets contain 5 elements, the combined set of scenar-

ios may contain 25 scenarios (5 contexts from f_a combined with 5 from f_b). It is obvious that for extremely complex flows, this back-propagation may produce a large number of scenarios, resulting in a combinatorial explosion, which is the drawback of the BP method. Given that $\mathrm{SIRL} = 5$ in this example, the collapsing requirement is met. Those 25 scenarios are collapsed into a single one containing the dummy flow f_X and its delay is set to maximum delay amongst the collapsed scenarios. When f finally progresses to n_2 and encounters the blocking flows f_c and f_d, the algorithm checks the history in its sequence $\{f_X, f\}$ and since there is no prior information regarding f_c and f_d, the analysis considers that these two flows are arriving for the first time and thus allows them to pass and interfere. The resulting scenarios would be $\{f_X, f_c, f_d, f\}$, $\{f_X, f_d, f_c, f\}$, $\{f_X, f_c, f\}$, $\{f_X, f_d, f\}$ and $\{f_X, f\}$. In contrast, BP would have retained the information regarding all scenarios, thus prohibiting the second interference caused by f_c and f_d, but at the expense of investigating $25 \times 5 = 125$ scenarios.

This example clearly shows (i) how loss of past information can reduce the chances of pruning infeasible scenarios and (ii) how the number of scenarios to be explored significantly decreases with the decease in the SIRL parameter – and more specifically in loaded networks in which the context of large scenarios have to be back-propagated to higher level nodes and combined to evolve the final scenarios. Conceptually, an SIRL set to ∞ implies BPC = BP while at the other end, an SIRL set to one tends towards obtaining the WCTT using the method proposed by [Ferrandiz et al. 2009], where no information about the past occurrence of any flow is retained. This approach is provided to the designer to handle the comparably small number of cases in which the BP algorithm may be inefficient. To formally validate the flexibility this offers, it will be seen in the experiment sections how lower SIRL will compute bounds tending towards those computed by [Ferrandiz et al. 2009], while with higher SIRLs we will have tighter WCTTs computed by BP. With this parameter, the system designer has the flexibility to trade-off computation time vs. pessimism in the computed WCTT.

8.3. Proof of Safety of "Branch, Prune and Collapse" (BPC) algorithm

In this section, we explain why the BPC method, by discarding some history information upon reaching the SIRL threshold, may output more pessimistic WCTT estimates compared to BP, but will under no circumstance lead to an unsafe WCTT estimate.

Let us denote by wcs the scenario (flow sequence) leading to the WCTT at run time which is *by definition a feasible scenario*. In order to prove that BPC is safe, we must prove that the BPC method does not eliminate this scenario wcs from the set of investigated scenarios – the method should never return a WCTT which is lower than that corresponding to the traversal time of wcs.

Firstly, it should be noted that, if we disable the two pruning mechanisms at lines 21 and 22 of Algorithm 2 and if SIRL is set to ∞, then our BPC algorithm boils down to an exhaustive enumeration of all possible scenarios at each router, and thus considers all possible blocking scenarios in the context of the analyzed flow (brute-force approach which is inherently safe). The pruning mechanisms of lines 21 and 22 use the precedence and time stamp information to identify some infeasible flow sequences and reduce the list of scenarios that need to be explored and facilitates the objective of obtaining tighter WCTTs. By definition, wcs is a feasible flow sequence and therefore, it will not be eliminated by these pruning techniques.

Given the loss of history information, the BPC method is unable to identify as many infeasible scenarios as the BP method. These infeasible scenarios (flow sequences) have flows that actually cannot occur at run time due to the task properties and hence add to extra-delays and bloat up the traversal time. Eventually, the set of scenarios explored will consider the set of feasible scenarios which includes wcs but will also include some infeasible scenarios, which are not identified due to loss of previous history and precedence information. Finally, on taking the maximum traversal time of all the scenarios, the method will return a value which is higher than or equal to the WCTT corresponding to wcs – hence the method is safe.

To summarize, BP investigates all feasible (including the wcs) and infeasible scenarios and prunes *all infeasible scenarios* by retaining the history information, and thereby is safe. BPC investigates all feasible and infeasible scenarios and prunes *some of infeasible scenarios* and as a consequence is still safe but more pessimistic.

8.4. Proof of termination of Algorithm GetContexts (Algo. 2)

The system is modeled as a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ with finite sets of nodes and bi-directional links and a set $\mathcal{F}$ of flows. Let $\mathcal{S}$ be the set of pairs $\langle f, \ell \rangle$, with $f \in \mathcal{F}$ and $\ell \in \mathcal{L}$, such that $\langle f, \ell \rangle \in S$ if and only if $\ell \in \mathrm{path}(f)$. Since $\mid \mathcal{L} \mid$ and $\mid \mathcal{F} \mid$ are finite, it holds that $\mid \mathcal{S} \mid$ is finite as well. A progress of a flow f from a link ℓ to a subsequent link ℓ' on its path is equivalent to the progress from the pair $\langle f, \ell \rangle$ to the pair $\langle f, \ell' \rangle$. If a flow f' blocks flow f on the link ℓ, it corresponds to the progress from the pair $\langle f, \ell \rangle$ to the pair $\langle f', \ell \rangle \in \mathcal{S}$. For a given flow f and a current link ℓ, the algorithm progresses in a forward manner to the next link $next(f, \ell)$ in the path of the flow f by invoking the function $\mathrm{getContexts}()$ at lines 8 and 24. Starting from any pair $\langle f, \ell \rangle \in \mathcal{S}$ (i.e. with f and ℓ as input), our algorithm investigates all the pairs, i.e. set of inputs, $\langle f', next(f', \ell) \rangle$ with $f' \neq f$ as a consequence of the round-robin arbitration policy. Then, the algorithm repeats the same (in a recursive manner) for each of these pairs $\langle f', next(f', \ell) \rangle$ and, as a consequence of the deadlock-free property of XY routing, we know that the initial pair $\langle f, \ell \rangle$ will never be re-visited. Additionally, since all the explored contexts are popped in line 20, the queue of pending scenarios is emptied and the algorithm eventually terminates.

9. SIMULATIONS AND RESULTS

We conducted several experiments with the dual objectives of comparing our method with the approach in [Ferrandiz et al. 2009] and studying the impact of varying different parameters on the WCTT of analyzed tasks. The simulation parameters have been summarized in the following table:

Network Size	8*8 mesh
Routing and switching mechanism	XY Routing, round robin arbitration
Router switching delay and transfer delay	1ns and 3ns (in-line with SCC [Intel 2010]))
Packet size and channel capacity	512 bytes, 1 Gbps
Implementation platform details	Intel dual-core desktop & Java (Max heap-size:4GB)

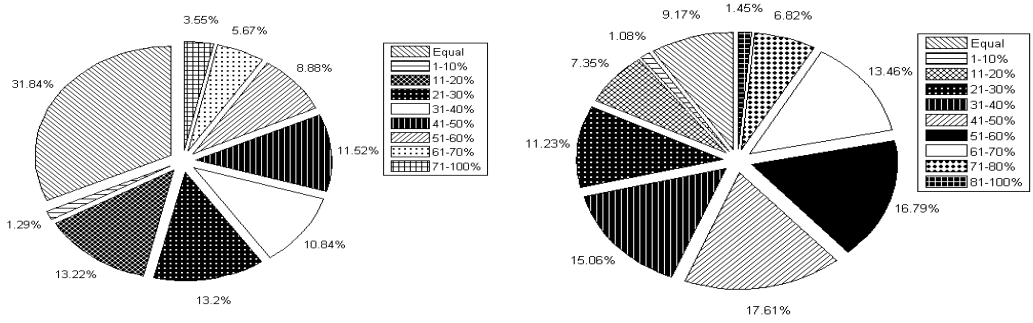
9.1. Comparison of BPC with the Approach of Ferrandiz et. al

As the improvements cannot be quantified in the general sense, since they are highly flow-set specific, we performed experiments on a wide range of different flow-sets in order to understand the trends and the ranges of improvement achieved by employing the proposed approach.

Test 1 (Network with moderate number of flows): We generated 200 random flow-sets, each having 64 flows. The flows originate from each tile but terminate at a random destination. The minimum inter-arrival time is a randomly generated parameter, varying between 5 to 20 microseconds. We computed the upper-bounds on WCTT of each flow using both the approaches and compared the results. For our approach, we selected a $\mathrm{SIRL} = 10000$.

In order to quantify the range of improvements, we computed a metric which we refer to as the "Percentage Improvement Ratio" (PIR) given by $(d_U - d_O^{10000}) * 100/d_U$, where d_U denotes the upper-bound on WCTT returned by the approach in [Ferrandiz et al. 2009] which we also refer to as *unoptimized wctt*, and d_O^{10000} is the value returned by our algorithm for $\mathrm{SIRL} = 10000$, which we call *optimized wctt*. Therefore a $\mathrm{PIR} = 25\%$ implies that our approach provided 25% lower (i.e. tighter) WCTT upper-bound.

Figure 7a summarizes our findings. We observed that for 31.84% of the flows, the bounds computed by both methods are equal, that is $d_O^{10000} = d_U$ and $\mathrm{PIR} = 0\%$. We have also demonstrated the percentage of flows and the improvement they achieved, in order to provide a deeper insight into the performance of our algorithm. As evident from Figure 7a, 1.29% of the flows had a PIR in the range $(1 - 10\%)$, 13.22% in the range



(a) Network with moderate number of flows (b) Network with high number of flows

Fig. 7. Distribution of WCTT improvement on the flows. The legends represent improvement ranges.

$(11 - 20\%)$ and so on. At the high end of the PIR scale, 3.55% of the analyzed flows returned $71 - 100\%$ tighter WCTT bounds.

The WCTT parameter*: If the computation terminates with the number of investigated scenarios not exceeding SIRL (implying that no collapses occur during the entire flow analysis), the method returns a value of the traversal time as would be computed by BP – we denote this result by WCTT*. In other words, all possible scenarios were analyzed and the one inducing the highest worst-case delay was recognized. Conversely, in cases when collapses occur, the returned WCTT presents only an *upper-bound* on the worst-case delay, without any additional details on how tight that bound is. When viewed from that perspective, the approach in [Ferrandiz et al. 2009] presents a special case of the proposed approach where $\mathrm{SIRL} = 1$. Therefore, [Ferrandiz et al. 2009] returns WCTT* only when the number of investigated scenarios is equal to "1".

In the 31.84% of the flows for which both methods returned equal values of WCTT, for $3/4^{th}$ of them, (23.96% of all the flows), the recursive-calculus method was able to capture WCTT*, inferring that these scenarios were simple and triggered the investigation of only one scenario. Therefore, in these cases there was no further scope for improvement. For the rest of the 68.16% flows, our algorithm returned tighter estimates. Based on the experiments, we can say that our algorithm performed equally well or dominated the method proposed by [Ferrandiz et al. 2009]. Also, for the selected SIRL value, the proposed approach managed to capture WCTT* in 92.13% of the cases, inferring that any additional increase in SIRL would not provide significantly tighter WCTT bounds, but would require exponentially greater amount of time.

The offline analysis completed within 24 hours, averaging a little bit more than 7 minutes per flow-set (each with 64 flows). The most complex flow-set took around an hour for completion, suggesting that the execution times may vary drastically when applied to flow-sets with identical characteristics but different flow routes, sometimes even by a high order of magnitude due to increase in the indirect contentions.

Test 2 (Network with high number of flows): The main purpose of this test was to check the efficiency of our algorithm when applied to a network with higher number of flows. In this test, we again generated 200 random flow-sets, 128 flows each, with *two flows originating from each tile* and terminating at a random destination. The minimum inter-arrival time is randomly generated parameter varying between 25 and 250 microseconds. For all flows, the values of d_U and d_O^{10000} were computed and compared. The simulation completed in 5 days, averaging 36 minutes per flow-set, where the most complex consumed around 3 hours, demonstrating that our approach is scalable and applicable to practical scenarios involving hundreds of concurrent flows.

As in the previous test-set, the PIR metric is used here to quantitatively express the improvements of the proposed method over the recursive calculus method [Ferrandiz et al. 2009]. We observed that for 9.23% of the flows, no improvements were made ($\mathrm{PIR} = 0\%$). For most of the flows without improvement (8.11%) the method [Ferrandiz et al. 2009] managed to capture WCTT*, with the same conclusion that for these sim-

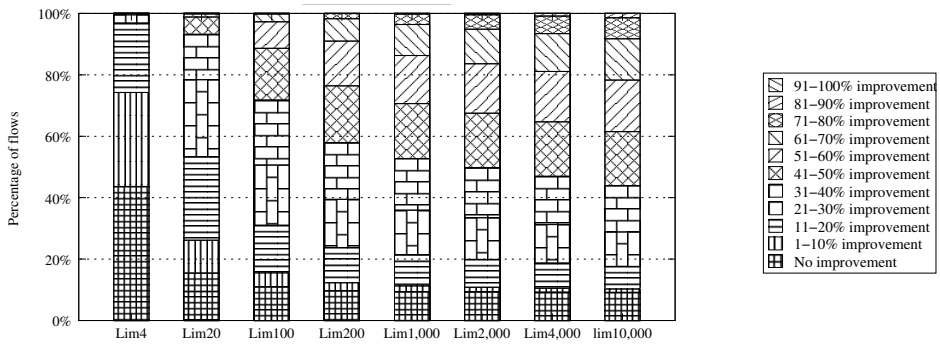


Fig. 8. Our proposed approach with varying SIRL vs. method by [Ferrandiz et al. 2009]

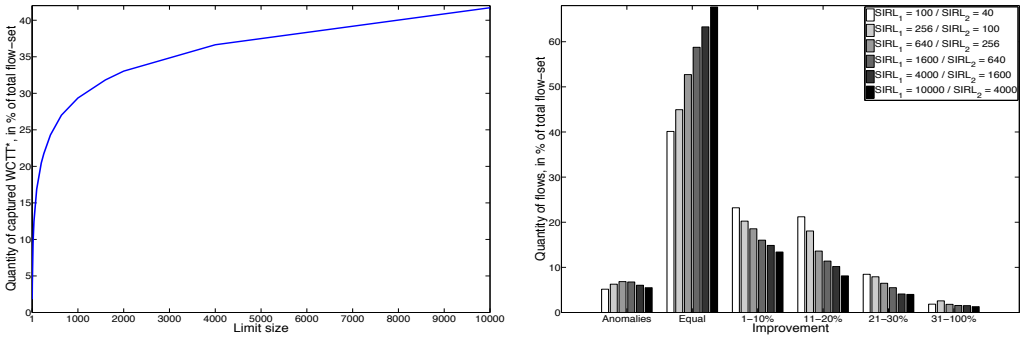
ple, one-scenario cases no improvements were possible. For the rest, i.e. 90.77% of the analyzed flows, it holds that $d_O^{10000} < d_U$, that is the upper-bound on the worst-case of the analyzed flows was tighter with our approach and the distribution is reflected in Figure 7b. It is interesting to see that more than 13% of the flows showed an improvement of $61 - 70\%$, while more than 8% of the flows show an improvement greater than 70%. Due to more complex traffic patterns resulting from increased amount of traffic, our approach with $\mathrm{SIRL} = 10000$ recognized WCTT* for 41.71% of the flows, which is significantly smaller when compared with the same of moderately loaded network. This suggests that the improvements can be achieved by increasing SIRL, but at the expense of additional computational complexity and memory consumption. Although the proposed approach takes a longer computation time, it clearly dominates the recursive calculus method in terms of obtaining tighter results. The selection of SIRL creates a trade-off between the computation time and the accuracy of the analysis.

Test 3 (Impact of SIRL on WCTT estimates) The objective of this set of experiment is to understand the impact of varying SIRL on the computed WCTT. The general intuition is that retaining more information about the scenarios provides more opportunities for eliminating invalid scenarios and therefore leads to tighter estimates. To validate this idea, we implemented our algorithm and executed it, by providing a different value of SIRL for each run and compared them against the results obtained by the approach in [Ferrandiz et al. 2009]. The results have been demonstrated in Figure 8 and like the previous experiments use the PIR metric for performance.

We observed that as the SIRL increases, the percentage of flows which show no improvements over the values computed by [Ferrandiz et al. 2009] decreases. Thus, with an $\mathrm{SIRL} = 4$, the WCTT computed for 43.6% flows exhibit no improvements, while with $\mathrm{SIRL} = 2000$ only 9.29% of the flows show no improvements (and the rest 90% of flows have tighter WCTTs). Note the marked shift in the distribution of improvements towards higher increased PIRs as the SIRLs increase. This is in accordance with the algorithm rationale that the retention of information about past flows in the scenarios can provide opportunities for tightening the WCTT. But as seen in the shift from 4000 to 10000, the PIR improvements do not differ much, as the opportunities for cutting down infeasible scenarios are exhausted. It can be then also inferred that choosing limits beyond a given SIRL will only burden the system memory of retaining information about those scenarios which may not lead to the WCTT. So a judicious decision must be taken by the system designer considering the tightness of results required and the time in which the tests must be performed.

Test 4 (Inter-SIRL ratios)

In the previous experiment, we compared the results of our approach with different SIRL values against the approach of [Ferrandiz et al. 2009]. In order to get a deeper insight into the impact of the SIRL parameter, we compared the results of our approach with different SIRL values against each other and plotted the results in Figure 9. The results coincide with the intuition, suggesting that greater values of SIRL improve the chances of capturing WCTT*, i.e. no collapses during the calculation occur (Figure 9a).



(a) Recognized worst-cases (WCTT*) across SIRLs (b) Improvements across SIRLs

Fig. 9. Inter-SIRL ratios

This claim is confirmed with a logarithmic growth in the number of non-collapsed scenarios across SIRLs.

Figure 9b shows that the relative improvements across SIRLs diminish as SIRL increases. That is, our method with $\mathrm{SIRL} = 100$ shows improvement against the same method with $\mathrm{SIRL} = 40$ in 60% of the cases, while the improvement is reported only in 30% of the cases when comparing results of $\mathrm{SIRL} = 10000$ and $\mathrm{SIRL} = 4000$. Thus, the number of scenarios with no improvement increases with SIRL. Conversely, the number of scenarios with improvements decreases with SIRL across all improvement ranges, suggesting that it may not be essential to perform the analysis with very high values of SIRL beyond a certain value. The benefits of analyzing with higher SIRL diminish as SIRL increases (especially for scenarios comprising of single-occurring flows).

As already stated, the value of the SIRL influences the frequency of scenario collapses. However, one interesting observation is the fact that higher SIRL does not necessarily always lead to a tighter WCTT upper-bound. We explain this with a following example. Consider the flow f in the example depicted by Figure 6. Let us assume that f_c and f_d are potential candidates for pruning. Now, assume that greater SIRL performs a collapse between occurrences of f_a and f_b. As the history information is lost, the flows f_c and f_d will contribute to the delays of both the flows, f_a and f_b. On the other hand, a smaller SIRL might trigger a collapse before (and after) the appearance of both f_a and f_b. In this case, it may successfully prune one appearance of f_c and f_d, thereby resulting in the situation (which we refer to an "anomaly") where a smaller value of SIRL returns a tighter WCTT estimate. As is visible from the results, the number of anomalies never exceeds more than 8% in all the considered cases.

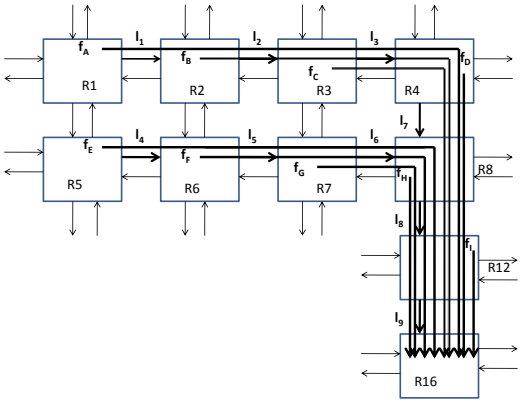


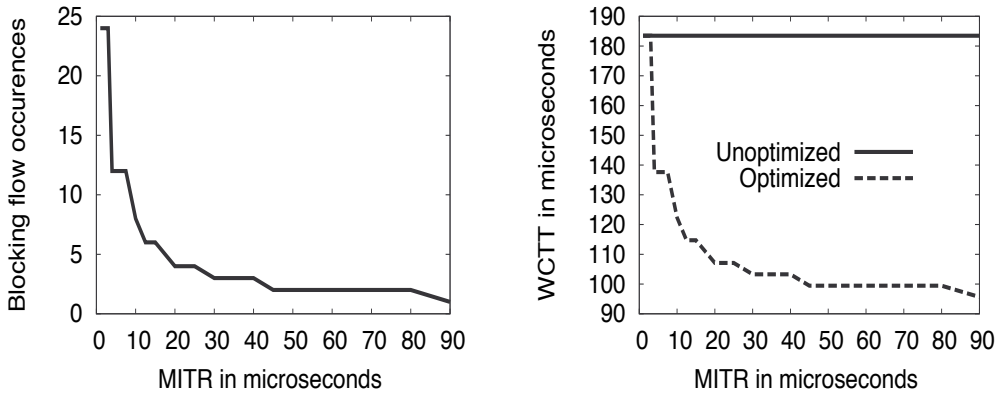
Fig. 10. Example Flow Set in a portion of the grid

9.2. Case Study with a specific flow-set

Figure 10 shows one of the flow-sets analyzed over a 4*4 grid that we shall use to demonstrate some interesting properties. Additional details like the core and cache engine were omitted to make the figure simpler. In the rest of the paper, we drop the prefix f and directly refer to the flow by its alphabetical name. In this flow set, we analyzed the flow A which originates in router R1 and terminates at the core associated with router R16. B, C, D, E, F, G, H and I are the other flows which also terminate at node R16. In this example, in the worst-case scenario, every flow can be potentially blocked by flow I (originating at $R12$) which is closest to the destination, then by flow H and so on. To provide a fair comparison with the approach in [Ferrandiz et al. 2009], we consider that every flow is by nature unregulated and non-blocking. By applying the approach in [Ferrandiz et al. 2009] or using our approach without any optimizations, one of the scenarios which resulted in the WCTT for flow A was:
$\{IHIEIDIHIEICIHIEIDIHIEIBIHIEIDIHIEICIHIEIDIHIEIA\}$.

This scenario is a manifestation of the "task-level" and "network level" pessimism and exemplifies the case of an over-estimated WCTT when infeasible blockings are not curtailed. Flows I and H are positioned in a manner which enables them to frequently block the other flows. This scenario also presents a useful example for exploring the delay on the WCTT of flow A if the task-profiles and task parameters of the blocking flows are varied.

9.3. Impact of Varying Packet Arrival Rates



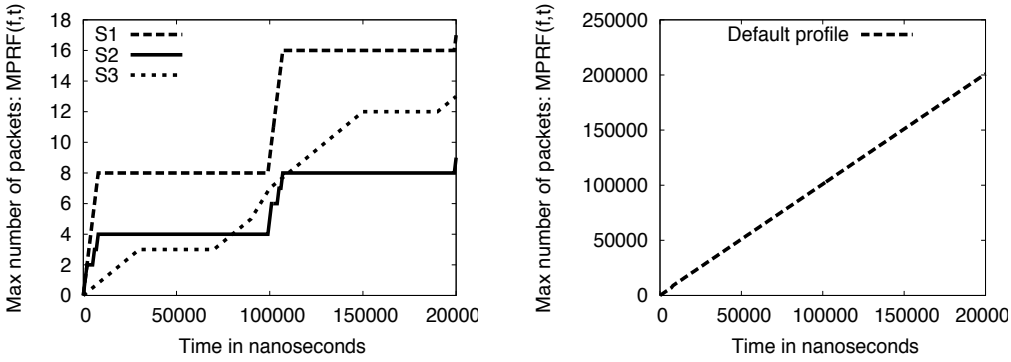
(a) No. of blockings by flow I vs. its $\mathrm{MinInterRel}(I)$ (b) WCTT of flow A vs. Packet Arrival rate of flow I

Fig. 11. WCTT of flow A decreases with increase in MIA time of flow I

The flows originating closer to the destination of the analyzed flow in Figure 10 have a higher tendency to block it directly and indirectly (by blocking the other flows which are also in the path). To verify this, we tuned the $\mathrm{MinInterRel}()$ (MIA in the figure) of flow I, increased it steadily and carried out the experiment, while keeping the nature of the other parameters constant. Figure 11a shows that as $\mathrm{MinInterRel}(I)$ increases, the number of times it can block the other flows is invalidated and thus the WCTT of flow A decreases as expected as seen in Figure 11b. In contrast, since the approach in [Ferrandiz et al. 2009] does not take into account the task characteristics, it allows these invalid blockings to progress and as a result, irrespective of the change in the flow parameters, the computed WCTT remains constant (see solid line in Figure 11b).

9.4. Impact of Varying Task Patterns

The WCTT of a packet of a given flow is also affected by the packet release profiles of the other flows. To study this effect, we conducted the following tests in which we analyzed the WCTT of flow A (in Figure 10) by changing the packet profiles of the blocking flows. In addition to the non-regulated non-blocking profile assumed for all



(a) Profiles S1, S2, S3 (b) Default profile

Fig. 12. $\mathrm{MaxPcktRel}()$ function applied to different profiles

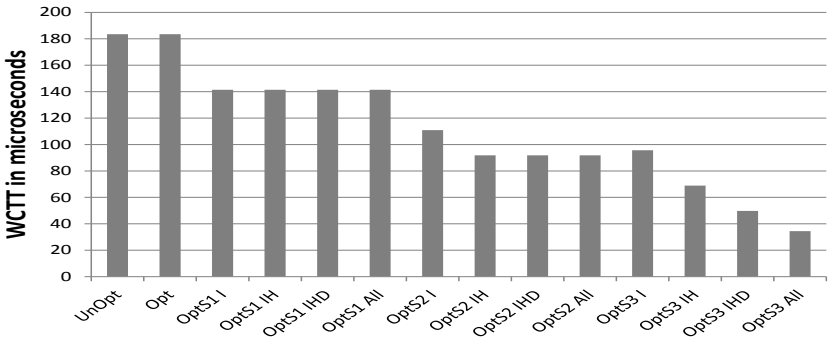


Fig. 13. WCTT of Flow A by varying the flow profiles of the blocking flows

the flows in [Ferrandiz et al. 2009], we defined three packet profiles S1, S2 and S3 (where S stands for Sparse) by generating synthetic pattern arrivals and computing the $\mathrm{MaxPcktRel}(f,t)$ for each of these profiles using the method proposed in [Dasari et al. 2011]. Figure 12 depicts the profiles, in which the X-axis represents the timeline (in nanoseconds) and the Y-axis shows an upper bound on the number of packets that can be generated in time t. The default profile, shown in Figure 12b, models the unregulated non-blocking profile. Figure 13 summarizes the results of different tests carried by varying the parameters of the blocking flows. The Y-axis represents the computed values of the WCTT of flow A. The X-axis contains the test name and should be read as $< approach >< profile >< flows >$, where $approach \in \{Unopt, Opt\}$ refers to the approach in [Ferrandiz et al. 2009] and our approach, respectively. All unmentioned flows by default have the profile presented in Figure 12b. So OptS2IH refers to a test case with our approach, the flows I and H have profile S2 and the other flows have the default profile.

As previously noted, the WCTT estimated by the Unopt method remained constant. Both the approaches computed the same WCTT for flow A for the default task profile. However, when we assigned the flows different profiles, Opt outperformed $Unopt$ in all the tests. This test case was designed to emphasize the importance of reducing the "task pessimism" described earlier. When applying the S1 profile to flow I, the WCTT of flow A reduced, since many occurrences of flow I were not feasible and were eliminated by the tests in our approach. We then applied the profile S1 to flows H and D, but this did not further impact the WCTT of flow A since they did not intercept flow A more than the admissible number of times. The effects of applying profile S1 can be observed in the WCTT values corresponding to tests OptS1I, OptS1IH, OptS1IHD and OptS1All in Figure 13. The S2 profile, by nature limits the generation of packets further and

additionally reduced the number of blockings of flows I and H. The profile S3 which is an extremely sparse packet profile, caused a major impact by drastically reducing the number of blockings and the resulting WCTT decreased further. The effects of applying profile S3 can be observed in the WCTT values corresponding to tests OptS3I, OptS3IH, OptS3IHD and OptS3All in Figure 13.

9.5. Comparison with Related work

The proposed approach considers the impact due to direct and indirect blockings as in [Lu et al. 2005]. Unlike the works proposed by [Diemer and Ernst 2010], [Shi and Burns 2008], [Paukovits and Kopetz 2008] or [Goossens et al. 2005], it does not need any hardware support for predictability, which is not commonly not present in existing platforms. Conversely our approach can be applied to a wider range of commercially available platforms. Furthermore, the proposed approach does not incur the delay of flushing out the preempted flits as done by pre-emption based techniques [Knauber and Chen 1999]. We make no assumptions on the state of the buffers as [Rahmati et al. 2009] and do not restrict the model to flows generating periodically arriving packets only. Due to the $\mathrm{MaxPcktRel}(f,t)$ function, even flows which generate packets randomly can be captured and thus our analysis is not restricted to affine-arrival curves as in [Qian et al. 2010]. Through experiments we have demonstrated that our approach computes tighter bounds when compared with the approach of [Ferrandiz et al. 2009], which assumes a model that is closely related to ours. Additionally, the idea of retaining the history information of flows to prune further infeasible flows lends novelty to our approach and the concept of having SIRL as a tunable parameter makes the approach flexible.

10. CONCLUSIONS

In this paper, we highlighted the problem of contention in a NoC as used in many-core architectures. We proposed a solution to compute the worst-case traversal time of a packet for such NoCs. This solution uses a branch and prune approach (BP) which improves on the work presented in [Ferrandiz et al. 2009] by leveraging the task characteristics and thereby provides tighter estimates on the computed WCTT. In order to tackle the complexity issues of BP in corner cases, we extended it to a branch, prune and collapse method (BPC), which via a configurable parameter provides a trade off between the computation time/memory usage and the WCTT tightness. A large set of experiments demonstrate the performance of the proposed algorithms in comparison with the approach of [Ferrandiz et al. 2009]. In particular, our work dominates their approach by yielding tighter WCTT estimates at the cost of extra computation time, the effects of which can be mitigated by the optimized version i.e. BPC. BPC on one hand limits the computational complexity, while on the other hand still provides the benefits of tighter WCTT bounds.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT and ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project Ref. FCOMP-01-0124-FEDER-022701; by FCT and COMPETE (ERDF), within REPOMUC project, ref. FCOMP-01-0124-FEDER-015050 ; by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grants SFRH/BD/71169/2010 and SFRH/BD/81087/2011.

REFERENCES

BENINI, L. AND DE MICHELI, G. 2002. Networks on chips: a new soc paradigm. *Computer 35,* 1, 70 –78.

BOUDEC, J.-Y. L. AND THIRAN, P. 2004. Network calculus - a theory of deterministic queuing systems for the internet. In *Lecture Notes in Computer Science, LNCS*. Springer Verlag.

DALLY, W. 1992. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems 3,* 2, 194 –205.

DALLY, W. AND SEITZ, C. 1986. The torus routing chip. *Distributed Computing 1*, 187 –196.

DASARI, D., ANDERSSON, B., NELIS, V., PETTERS, S. M., EASWARAN, A., AND LEE, J. 2011. Response time analysis of COTS-based multicores considering the contention on the shared memory bus. In *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. 1068–1075.

DIEMER, J. AND ERNST, R. 2010. Back suction: Service guarantees for latency-sensitive on-chip networks. In *Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. 155 –162.

DRAPER, J. T. AND GHOSH, J. 1994. A comprehensive analytical model for wormhole routing in multicomputer systems. *J. Parallel Distrib. Comput. 23,* 2, 202–214.

FERRANDIZ, T., FRANCES, F., AND FRABOUL, C. 2009. A method of computation for worst-case delay analysis on spacewire networks. 19–27.

FERRANDIZ, T., FRANCES, F., AND FRABOUL, C. 2011. Using network calculus to compute end-to-end delays in spacewire networks. *SIGBED Rev. 8,* 3, 44–47.

FERRANDIZ, T., FRANCES, F., AND FRABOUL, C. 2012. A sensitivity analysis of two worst-case delay computation methods for spacewire networks. In *Euromicro Conference on Real-Time Systems*. 47–56.

GOOSSENS, K., DIELISSEN, J., AND RADULESCU, A. 2005. Aethereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers 22,* 5, 414 – 421.

HU, J. AND MARCULESCU, R. 2003. Energy-aware mapping for tile-based noc architectures under performance constraints. In *8th Design Automation Conference*. 233–239.

INTEL. 2010. The single-chip-cloud computer. http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf.

KNAUBER, K. AND CHEN, B. 1999. Supporting preemption in wormhole networks. In *The Twenty-Third Annual International Computer Software and Applications Conference, COMPSAC*. 232 –238.

LEE, S. 2003. Real-time wormhole channels. *Journal Of Parallel And Distributed Computing 63*, 299–311.

LU, Z., JANTSCH, A., AND SANDER, I. 2005. Feasibility analysis of messages for on-chip networks using wormhole routing. 960–964.

PAUKOVITS, C. AND KOPETZ, H. 2008. Concepts of switching in the time-triggered network-on-chip. In *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 120 –129.

PELLIZZONI, R., SCHRANZHOFER, A., CHEN, J.-J., CACCAMO, M., AND THIELE, L. 2010. Worst case delay analysis for memory interference in multicore systems. In *Conference on Design, Automation and Test in Europe*. 741–746.

QIAN, Y., LU, Z., AND DOU, W. 2010. Analysis of worst-case delay bounds for on-chip packet-switching networks. *IEEE Transactions onComputer-Aided Design of Integrated Circuits and Systems 29*, 802 – 815.

RAHMATI, D., MURALI, S., BENINI, L., ANGIOLINI, F., DE MICHELI, G., AND SARBAZI-AZAD, H. 2009. A method for calculating hard QoS guarantees for networks-on-chip. 579 –586.

SALMINEN, E., A., K., AND T., H. 2008. *Survey of Network-on-Chip Proposals*. www.ocpip.org.

SCHLIECKER, S., NEGREAN, M., AND ERNST, R. 2010. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *Conference on Design, Automation and Test in Europe*. 759–764.

SHI, Z. AND BURNS, A. 2008. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip*. 161 –170.

TILERA. 2011. Tile processor: user architecture manual. www.tilera.com/scm/docs/UG101-User-Architecture-Reference.pdf.

WENTZLAFF, D., GRIFFIN, P., HOFFMANN, H., BAO, L., EDWARDS, B., RAMEY, C., MATTINA, M., MIAO, C.-C., III, J. F. B., AND AGARWAL, A. 2007. On-chip interconnection architecture of the tile processor. *IEEE Micro 27*, 15–31.