A Stochastic Response Time Analysis for Communications in On-Chip Networks

Meng Liu, Moris Behnam, Thomas Nolte
Mälardalen University, Västerås, Sweden
Email: {meng.liu, moris.behnam, thomas.nolte}@mdh.se

***Abstract*—Priority-based wormhole-switching has been proposed as a solution to handle real-time traffic in on-chip networks. In order to support real-time traffic, the predictability of end-to-end delays need to be guaranteed. Several deterministic schedulability analysis approaches for wormhole-switched networks have been proposed. These approaches calculate a single upper-bound of the response time of each Network-on-Chip (NoC) flow, which is suitable for hard real-time applications. However, for many soft real-time applications, the performance does not depend on the worst-case scenario, which means that the calculated single upper-bounds are not sufficient to represent the performance. Therefore, in this paper, we present a stochastic Response Time Analysis (RTA) which can calculate a distribution of the response times of a real-time NoC flow. The estimated distributions can be utilized for multiple purposes, such as calculating deadline miss ratios, and computing upper-bounds regarding different probabilities. A number of simulation-based experiments are generated in order to investigate the pessimism involved in the analysis. Moreover, the processing time of the analysis is also measured from the experiments in order to examine the scalability of the proposed approach.**

I. Introduction

Many-core platforms are gaining attention in industry due to its high computing capability along with limited hardware sizes. On such a platform, many components are integrated into a single chip. The chip with its components is known as a system-on-chip. The communication between different components in such systems is typically achieved by an on-chip network (also called Network-on-Chip (NoC)). Wormhole-switching (also known as wormhole-routing) is a technique widely used in most of the existing NoCs. Using wormhole-switching, each router in the network requires much less buffer size compared to the store-and-forward mechanism. This fulfills an important requirement of a NoC that the hardware size needs to be limited.

During the past decades, many research works have been done regarding running real-time applications on many-core platforms. In addition to the requirement of functional correctness, timeliness is one of the most important nonfunctional properties of a real-time application. The timing behavior of real-time applications should be predictable, so that the system designers can verify if the timing requirements can be satisfied or not. For a hard real-time application, all the timing requirements must be strictly fulfilled, otherwise the system may face catastrophic consequences (e.g. the brake-by-wire system or the airbag system in a car would fail). For a soft real-time system, some violations of the timing requirements are acceptable, which may only degrade the system performance without causing serious problems (e.g. a multimedia entertainment system would temporally deliver less than perfect pictures or sounds).

Therefore, several schedulability analysis approaches have been proposed for on-chip networks (e.g. [1][2][3]). Most of the approaches aim to calculate the worst-case end-to-end delay (also called Worst-Case Response Time (WCRT) hereinafter) of each NoC flow. Such analyses are more suitable for hard real-time applications, since the worst-case behavior affects the reliability of the system even though it may potentially only happen in very rare occasions. However, for soft real-time applications, it is often the case that the worst-case scenario is not sufficient to represent the performance [4]. For example, Active Safety functions in automotive systems (such as Lane Keeping, Lane Departure Warning, etc.) rely on the average response times which need to be controlled and minimized [5]. Therefore, most of the existing deterministic analyses cannot be directly utilized for such soft real-time applications.

In this paper, we present a stochastic Response Time Analysis (RTA) for wormhole-switched on-chip networks. The analysis computes a distribution of response times of each flow instead of a single upper-bound.

A. Contributions

This paper contains the following contributions:

- We present a stochastic RTA for wormhole-switched on-chip communications. The analysis calculates a distribution of response times of each NoC flow. The computed distribution can provide much more information compared to the Worst-Case Response Time (WCRT) calculated by a deterministic analysis, such as upper-bounds of response times regarding different probabilities, average response times, and Deadline Miss Ratio (DMR).
- A number of simulation-based evaluations are implemented in order to investigate the level of pessimism involved in the stochastic analysis. Moreover, the evaluation results also show the pessimism included in the deterministic RTA presented in [1][2], which is a property that has not been evaluated in these works.
- The processing time of the analysis during the experiments is measured in order to examine the scalability of

This work has been supported by the Swedish Research Council (Vetenskapsrådet) through the project START.

the proposed approach.

B. Related Work

The wormhole switching technique was introduced in academia more than 20 years ago [6]. Later on, this technique was utilized in the Network-on-Chip architecture [7] because of its high throughput along with a small buffer requirement [8]. In order to use platforms equipped with on-chip networks for real-time applications, the predictability of the timing behavior of NoCs needs to be guaranteed. A number of research works focusing on providing predictable NoCs have been proposed in the literature, such as the Back Suction flow-control scheme [9], the Time-Triggered Network-on-Chip (TTNoC) [10], and the Æthereal NoC [11].

In order to verify if the timing requirements of a NoC can be fulfilled, a schedulability test is necessary. In [12], the authors present a Network Calculus [13] based analysis approach along with a Recursive Calculus to compute the end-to-end transmission delays of flows in a wormhole-switched network using the Round-Robin arbitration policy. This work is extended in [3] to a more general approach called Branch, Prune and Collapse. In [1], the authors present a deterministic Response Time Analysis (RTA) for priority-based on-chip networks, which is based on the well-known response time analysis for task scheduling [14]. The analysis proposed in [1] assumes that each NoC flow is assigned a distinct virtual-channel. However, more virtual-channels require more memory on each router, which makes the assumption unrealistic for most of the existing commercial NoC implementations. Therefore, the authors in [2] present an extended RTA to handle NoC flows with shared priorities. In this work, one virtual-channel is assigned to each priority level, and multiple flows can share the same priority/virtual-channel. Most of these existing analyses are deterministic approaches, which means that they are used to compute a single upper-bound of the response times of each NoC flow. This is suitable for hard real-time applications, since the analysis results are guaranteed. However, for soft real-time applications, the information of the WCRTs is not sufficient to verify the fulfilment of requirements. Therefore, we present a stochastic RTA for wormhole-switched on-chip networks, which can compute a probability distribution of response times of each NoC flow. Our work targets priority-based NoCs with soft real-time constraints.

A number of stochastic timing analyses targeting real-time task scheduling have been proposed in the literature. In [15], the authors introduce a stochastic time demand analysis to calculate a lower bound of the deadline meeting ratio of each task in a system using fixed priority scheduling. Diaz et al. present a probabilistic analysis in [16], which can be used for both static and dynamic priority based scheduling. In this work, the execution time of each task is represented by a probability distribution instead of a single value. Later on, this analysis is applied on Control Area Networks (CAN) in [4], where the random sampling delays are taken into account in addition to stochastic packet sizes. In [17], the authors propose another extension of the analysis from [16] to support tasks with multiple probabilistic parameters, where the inter-arrival times between successive instances of the same task can also be stochastic. The stochastic RTA proposed in this paper is also based on the work presented in [16], where we take into account the features of wormhole-switched NoCs.

The rest of this paper is organized as follows. In Section II, we recapitulate the probabilistic RTA for tasks. In Section III, we briefly describe the system model used in this paper. The details of the proposed stochastic RTA for NoCs are explained in Section IV. Results of simulation-based evaluations are discussed in Section V. Finally, Section VI concludes the paper along with thoughts about the future work.

II. Stochastic RTA for tasks

We first recall the stochastic time analysis for tasks presented in [16][18], which is the basis of our work. In this analysis framework, the Worst-Case Execution Time (WCET) of each task τ_i is described as a random variable with a discrete probability distribution $\mathcal{C}_i$[1].

The analysis in [16] assumes that all the tasks are periodic with deterministic release times. Therefore, for a set of $i-level$ tasks (i.e. all the tasks with priority higher than or equal to the priority of τ_i), the activation pattern of one hyperperiod will be repeated in all the other hyperperiods. As a result, the analysis mainly depends on the *backlog* (i.e. the workload from tasks with higher or equal priority of τ_i which are generated before a certain time instant Δt and still not completed at Δt) at the beginning of each hyperperiod. Díaz et al. show that this backlog is a random variable with a certain distribution, but the sequence of these random variables is a Markov chain. Once the stationary $i-level$ backlog distribution is computed, the response time distribution of τ_i can be calculated.

If the total task utilization (i.e. the maximum system utilization) is smaller than 1, the maximum generated workload cannot exceed the length of a hyperperiod. As a result, the backlog at the end of each hyperperiod can be bounded. Moreover, we can also observe that if all the tasks have at least one job released in the first hyperperiod, the backlog at the end of this hyperperiod will be repeated at the end of all the following hyperperiods. In this case, in order to obtain the stationary backlog distribution, we only need to compute the backlog at the end of the first hyperperiod where all the tasks are released at least once.

Then the analysis computes the response time PMF for all the jobs (i.e. task instances) of τ_i in one hyperperiod, where the stationary $i-level$ backlog is utilized at the beginning of the hyperperiod. The response time PMF of a job $\tau_{i,j}$ ($j = 1,2,3,...$)[2] is computed as

$$\mathcal{R}_{i,j} = \mathcal{B}_{i,j}(\lambda_{i,j}) \otimes \mathcal{C}_i \otimes I_{i,j}(\lambda_{i,j}) \quad (1)$$

where $\mathcal{B}_{i,j}$ denotes the backlog of tasks with higher priorities than τ_i which are released before time instant $\lambda_{i,j}$ and still not completed at $\lambda_{i,j}$. $\mathcal{C}_i$ represents the execution time PMF of

[1] In this paper, we use calligraphic letters to represent discrete probability distributions. Moreover, we use a *Probability Mass Function (PMF)* to represent each distribution hereinafter.

[2] We use the second subscript to represent the index of the job, i.e. $\tau_{i,j}$ denotes the jth job of τ_i.

τ_i. $I_{i,j}(\lambda_{i,j})$ denotes the interference caused by higher priority tasks which are released at or after time instant $\lambda_{i,j}$. Eq. 1 is solved iteratively using two operations called *convolution* and *shrinking*.

The convolution operation is denoted by $\otimes$, which is used to sum up two distributions. An example of this operation is shown in Example 1.

Example 1.

$$\begin{array}{ll} & \binom{1\ \ 2}{0.5\ \ 0.5} \otimes \binom{2\ \ 3}{0.3\ \ 0.7} \\ = & \binom{1+2\ \ 1+3\ \ 2+2\ \ 2+3}{0.5\times0.3\ \ 0.5\times0.7\ \ 0.5\times0.3\ \ 0.5\times0.7} \\ = & \binom{3\ \ 4\ \ 5}{0.15\ \ 0.5\ \ 0.35} \end{array}$$

The shrinking operation is used to simulate time progress, and can be denoted by $SHRINK(\mathcal{X},t)$. If $\mathcal{X}$ is the distribution of a certain workload at the beginning of a time duration t, the result of $SHRINK(\mathcal{X},t)$ is a distribution of the remaining workload at the end of t. The operation can be presented as

$$\begin{array}{ll} SHRINK(\mathcal{X},t) \Rightarrow & \binom{x_k}{Pr(X=x_k)} = \binom{MAX(x_k-t,0)}{Pr(X=x_k)}, \\ & for\ \forall \binom{x_k}{Pr(X=x_k)} \in \mathcal{X} \end{array} \quad (2)$$

where $Pr(X=x_k)$ denotes the occurrence probability of $X=x_k$. An example is given in Example 2.

Example 2.

$$\begin{array}{ll} & SHRINK(\binom{1\ \ 2\ \ 3\ \ 4}{0.5\ \ 0.2\ \ 0.2\ \ 0.1},\ 2) \\ = & \binom{1-2\ \ 2-2\ \ 3-2\ \ 4-2}{0.5\ \ \ 0.2\ \ \ 0.2\ \ \ 0.1} \\ = & \binom{0\ \ 1\ \ 2}{0.7\ \ 0.2\ \ 0.1} \end{array}$$

Apparently, the amount of workload cannot be negative. When the amount of the workload is less than the time duration (e.g. the first value of the PMF is 1, which is less than the time duration 2), there will be no remaining workload at the end of time duration t. Therefore, in the result of this example, the remaining workload has a probability of 0.7 to be 0.

The calculation of Eq. 1 starts with the time instant that is the beginning of a hyperperiod. In the case that task τ_i is released with its critical instant [19] (i.e. $\tau_{i,j}$ is released together with all the other tasks with higher or equal priority), the initial backlog can be computed as $\mathcal{B}_{i,j}(0) = \bigotimes_{p\in hp(i)} \mathcal{C}_p$. Then the analysis will continuously check the workload at the release times of the following jobs with higher or equal priorities. Assume that a job of a higher priority task τ_p is released at time t_k (where $k=1,2,...$), $\mathcal{B}_{i,j}(t_k)$ can be computed by shrinking the workload PMF at t_{k-1} to t_k. Then the workload at t_k can be calculated by the convolution of $\mathcal{B}_{i,j}(t_k)$ and $\mathcal{C}_p$. By repeating the above step, the tail of the distribution of $\mathcal{R}_{i,j}$ will be iteratively revised. The calculation of Eq. 1 terminates (1) when the remaining workload becomes 0, which means that there is no more interference caused to $\tau_{i,j}$; or (2) when the release times of newly arrived interference are later than the deadline.

The response time PMFs of all the jobs of τ_i in a hyperperiod need to be computed. Finally, the response time PMF of τ_i equates to the average of all these PMFs. Unfortunately, when the hyperperiod of a set of tasks is long, a large amount of task instances may need to be taken into account during the analysis. As a result, the complexity of the whole analysis can become high. In order to reduce the time and memory cost of the analysis, several approximation solutions are proposed.

First, we need to introduce the definition of the *'greater than'* relationship between two probability distributions.

Definition 1. *[18] Let $\mathcal{X}$ and $\mathcal{X}'$ be two random variables with different distributions. We say that $\mathcal{X}$ is greater than $\mathcal{X}'$ (denoted by $\mathcal{X} \succeq \mathcal{X}'$), if $Pr(X \le x) \le Pr(X' \le x)$ for any x.*

Fig. 1 visually illustrates the 'greater than' relationship between two distributions. If $\mathcal{X}$ is greater than $\mathcal{X}'$, the curve showing the Cumulative Distribution Function (CDF) of $\mathcal{X}$ should be below the curve of $\mathcal{X}'$. In this example, $\mathcal{X}$ and $\mathcal{X}''$ are not comparable. From the stochastic analysis point of view, if $\mathcal{X}'$ represents the actual response time distribution of a certain task, a greater distribution $\mathcal{X}$ can be considered as a pessimistic response time distribution of the same task. For example, given a specific value x, in the distribution of $\mathcal{X}$, the variable has a lower probability to exceed x. On other hand, given a certain exceedance probability, the estimated bound in $\mathcal{X}$ is larger than the one in $\mathcal{X}'$ (see the horizontal line at the probability 0.5 in Fig. 1).

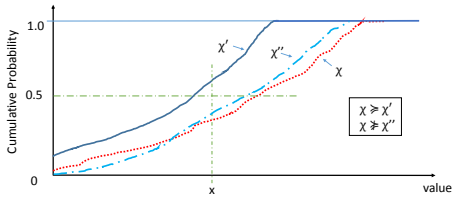

Fig. 1. An example showing the 'greater than' relationship between two distributions.

When we use approximations to reduce the calculation complexity, we need to guarantee that the analysis results are safe (i.e. can be pessimistic but not optimistic). Therefore, in this paper, we use a theorem proved in [17].

Theorem 1. *We consider a task set with n tasks, where the execution time of each task is characterized by a random variable. The task set is scheduled preemptively on a single processor. The response time distribution $\mathcal{R}_{i,1}$ of the first job of task τ_i (i.e. released at its critical instant [14] which can result in the worst-case scenario) is greater than the response time distribution of any other job of τ_i.*

This theorem implies that if the response time distribution of a task is constructed by analyzing the instance which is released at the critical instant, the result is safe but pessimistic.

III. System Model

Before presenting our stochastic time analysis for NoC, we first describe the system model assumed in this paper. The system under consideration is a many-core platform with a 2D-meshed wormhole-switched on-chip network, which consists of $m \times m$ tiles. In this paper, we assume that each tile (also called *node* hereinafter) consists a single core and a router. Each two adjacent nodes are connected by full-duplex links, and each link can support multiple virtual-channels (the same

as assumed in [1][2]). The bandwidths of all the links are assumed to be identical. An example is presented in Fig. 2, where the platform consists of 16 cores connected by a 2D-meshed NoC.

The network comprises a set S of n periodic or sporadic real-time message-flows (i.e. $S = (f_1, f_2, ..., f_n)$). Each flow consists of infinite instances (also called *packet* hereinafter), and can be characterized as $f_i = (\mathcal{L}_i, T_i, D_i, P_i, \Re_i)$. Instead of using a single upper-bound for the message size of each flow as assumed in [1][2][20], we use a discrete probability distribution $\mathcal{L}_i$ to represent the possible message sizes[3] $\mathcal{L}_i$ of each flow f_i.

$$\mathcal{L}_i = \begin{pmatrix} l_{i,1} & l_{i,2} & \cdots & l_{i,N_i} \\ Pr(L_i{=}l_{i,1}) & Pr(L_i{=}l_{i,2}) & \cdots & Pr(L_i{=}l_{i,N_i}) \end{pmatrix} \quad (3)$$

where $N_i \in N^+$. In other words, $\mathcal{L}_i$ comprises N_i random values together with the corresponding probabilities. If we set the probability $Pr(L_i = l_{i,k}) = 1$, the message model will become the same as the deterministic model with a single upper-bound $l_{i,k}$. T_i denotes the period or minimum inter-arrival time between two successive packets of flow f_i. D_i denotes the relative deadline of each flow. In this paper, we assume that each flow has a constrained deadline (i.e. $D_i \leq T_i$). P_i represents the priority of f_i. $\Re_i$ denotes the fixed path/route of flow f_i. Moreover, in this paper, we do not consider any specific release offset of each flow.

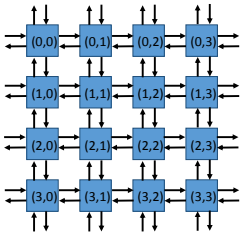

Fig. 2. An example of a 4×4 many-core platform with meshed NoC.

The following notions are used in the rest of the paper.

- S: the flow set including all the flows in the network
- S_i^D: the flow set including all the flows that can cause direct interference to f_i
- S_i^I: the flow set including all the flows that can cause indirect interference to f_i
- S_i^B: the flow set including all the flows that can cause blocking to f_i due to sharing of a virtual-channel
- $L \leftarrow a$: adding an item a into a list L
- $L \rightarrow a$: removing an item a into a list L

IV. Stochastic RTA for NoC

In this section, we present the stochastic RTA for on-chip networks based on the approach discussed in Section II. In fact, from the analysis point of view, scheduling tasks and message flows are similar, which can both be modeled as applications sharing certain resources (e.g. multiple tasks sharing one processor or a number of messages sharing one communication bus). When the access to a certain resource is limited, the applications have to wait in a queue for accessing the resource.

In order to extend the stochastic RTA for tasks [16] to an analysis of wormhole-switched NoCs, we need to consider the main differences of the system behavior between these two frameworks.

- The work in [16] is based on completely preemptive scheduling. However, a wormhole-switched NoC uses a flit-level preemptive mechanism. Therefore, the analysis needs to take a blocking delay into account (details in Section IV-B).
- In an on-chip network, even though a higher priority flow f_j does not share any link with the flow under analysis f_i, f_j can still affect the response time of f_i. This is called the effect of *indirect interference* (more details in Section IV-C). This behavior cannot occur in the framework assumed in [16].

A. Basic Transmission Latency PMF

First, we compute the basic transmission latency (i.e. the network latency without any interference or blocking from other flows) for each flow. In fact, the basic transmission latency of a NoC flow is similar to the concept of the execution time of a task in the context of task scheduling. Since we use a distribution $\mathcal{L}_i$ to represent packet sizes, the corresponding basic transmission latencies are also depicted by a probability distribution $\mathcal{C}_i$.

The transmission latency for a packet consists of the propagation delay through the whole path and the processing delay inside each passed router. Therefore, the basic network latency for a packet of f_i with the size of $l_{i,k}$ ($k \in [1, N_i]$) can be computed by

$$c_{i,k} = \lceil \frac{l_{i,k} - f_{size}}{f_{size}} \rceil \cdot \frac{f_{size}}{B_{link}} + nhops(\Re_i) \cdot (d_s + \frac{f_{size}}{B_{link}}) \quad (4)$$

where f_{size} denotes the size of a single flit, B_{link} is the bandwidth of the links, $nhops(\Re_i)$ represents the number of hops of the route $\Re_i$, and d_s symbolises the processing delay inside each router.

Since the computation of $c_{i,k}$ is based on a specific packet size $l_{i,k}$, the occurrence probability of $c_{i,k}$ (i.e. $Pr(C_i = c_{i,k})$) equates to $Pr(L_i = l_{i,k})$. Accordingly, by computing transmission time regarding all the values of C_i, we can get the PMF of the basic transmission latency for flow f_i as

$$\mathcal{C}_i = \begin{pmatrix} c_{i,1} & c_{i,2} & \cdots & c_{i,N_i} \\ Pr(C_i{=}c_{i,1}) & Pr(C_i{=}c_{i,2}) & \cdots & Pr(C_i{=}c_{i,N_i}) \end{pmatrix} \quad (5)$$

where $N_i \in N^+$. The basic transmission latency PMF for each task will be used in the stochastic RTA analysis.

B. Blocking Delay

Blocking delay of f_i is caused by the non-preemptive transmission of a flow with lower priority. As mentioned in Section III, we assume that the network under analysis can support flit-level preemptions. In other words, a preemption cannot occur during the transmission of a single flit, but can occur between flits. Therefore, a packet can at most experience a blocking delay due to the transmission of one single flit from

[3]The message size $\mathcal{L}_i$ includes all the necessary packet segments such as header, payload and tail flit.

a lower priority flow at each hop. The blocking delay of a flow f_i can then be upper-bounded by

$$b_i = nhops(\Re_i) \times (\frac{f_{size}}{B_{link}} + d_s) \quad (6)$$

In order to reduce the analysis complexity, we use an upper-bound instead of a probability distribution for the blocking delay of each packet. This approximation can cause pessimism in the results. However, for many applications, the pessimism is quite small, because the small flit size results in a very low blocking delay which is much less than the response time of a packet. In many NoC products, the flit size matches the width of the physical channel, which is typically less than 256 bits (e.g. 160 bits in a Tilera chip [21], and 256 bits in the Intel Sandy Bridge microprocessor [22]).

C. Indirect Interference

Apparently, two flows can affect the response times of each other when they need to compete for the access of the same links. However, as discussed in [1], a flow f_m may affect the response time of f_i even if they do not share any links. In this case, f_m is considered to cause *indirect interference* to f_i.

Fig. 3-a shows an example of the indirect interference behavior. Assume that f_i is the flow under analysis with the lowest priority. f_j and f_i share links between Node-B and Node-C, and f_j has priority higher than that of f_i. Therefore, f_j can cause interference to f_i. This type of interference due to competing for the same resource is called *direct interference*. Assume that f_m and f_j share links between Node-A and Node-B, and that f_m has priority higher than that of f_j. In this case, f_j can get direct interference from f_m before f_j reaches Node-B. As a result, the arrival pattern of f_j at Node-B is no longer periodic. As shown in Fig. 3-b, the inter-arrival time between two instances of f_j can become smaller than T_j. Therefore, when we calculate the response time of f_i, we cannot simply consider f_j as a periodic flow with a fixed period of T_j which can incur optimistic results. As discussed in [1], while analyzing f_i, the direct interference of f_j before Node-B can be considered as an extra jitter of f_j which is called *interference jitter* (i.e. J_j^I).

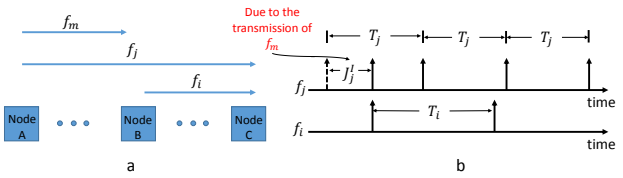

Fig. 3. An example showing the indirect interference behavior.

If the interference jitter of f_j is a random variable, the arrival pattern of f_j can also be considered as random while computing a probabilistic response time of f_i. However, this consideration can lead to a fast growth of computation complexity as the number of higher priority flows goes up. Assume that there are n flows causing direct interference to f_i, and each of these flows contain k possible arrival patterns. Then the arrival pattern considering all these flows can include k^n combinations. In order to reduce the analysis complexity, we use an approximation based on a single upper-bound of the interference jitter of each flow. In this case, *the critical instant of f_i can be constructed as the time instant when f_i arrives at the shared path together with all the other higher priority flows who share links with f_i. The first instance of any of these higher priority flows experiences its maximum interference jitter and the following packets arrive as early as possible.*

As discussed earlier in the example, the interference jitter of f_j is caused due to the transmission of f_m. In other words, J_j^I is incurred by the direct interference that f_m causes to f_j. Similarly, all the flows, which can cause direct interference to f_j, can contribute to J_j^I. Therefore, the interference jitter of f_j can be upper-bounded by $R_j - C_j$ [1], where R_j and C_j respectively represent the worst-case response time and the basic transmission delay of f_j. In our analysis, R_j and C_j are denoted by probability distributions. Accordingly, the upper-bound of J_j^I can be modified as $R_j^{max} - C_j^{min}$, where R_j^{max} is the maximum value of $\mathcal{R}_j$, and C_j^{min} denotes the minimum value of $\mathcal{C}_j$. Moreover, as discussed in [1], the interference jitter J_j^I exists if and only if $S_j^D \bigcap S_i^I = \varnothing$. Therefore, we can compute the interference jitter of f_j as

$$J_j^I = \begin{cases} R_j^{max} - C_j^{min} & , \; if \; S_j^D \bigcap S_i^I = \varnothing \\ 0 & , \; else \end{cases} \quad (7)$$

D. Stochastic RTA of a NoC without Priority Sharing

In this section, we present the stochastic RTA of a NoC without priority sharing, where we take into account the effects caused by the system behaviors discussed in the previous sections.

Algorithm 1 presents the stochastic RTA, which can calculate a probability distribution of the response times of each NoC flow as results. Similar to the approach presented in Section II, the analysis starts from time instant 0, and iteratively takes into account the new arrivals of all the packets which can contribute to the response time of the flow under analysis (i.e. f_i).

First of all, we need to create a list of check-points CP_i (line 2, Alg. 1) for f_i. Each check-point represents a time instant when a new packet arrives. The creation process of CP_i is presented in Alg. 2. Because of the use of a priority-based arbitration mechanism, we only need to consider the new arrivals of higher priority flows who share links with f_i (line 2, Alg. 2). We assume that the network employs an *abort on deadline* policy, which means that the packet will be discarded once it misses its deadline. From the scheduling point of view, this policy can ensure that one deadline miss of a packet will not cause multiple deadline misses of the following packets. From the perspective of applications, this policy can prevent misuse of expired data. Under this mechanism, we need to consider the arrivals of a higher priority flow f_j within the time window of $[0, D_i]$. As discussed in Section IV-C, the indirect interference can affect the response time of f_i by decreasing the packet inter-arrival times of flow f_j which can cause direct interference to f_i. The effect of the indirect interference is transformed to an extra jitter of f_j, and the upper-bound of this jitter J_j^I can be computed by Eq. 7. Similar

to the solution discussed in Section II, it is safe to perform the analysis focusing on the flow instance which is released with its critical instant. As discussed earlier, the critical instant of f_i is the time instant when f_i is released together with a packet of f_j with its maximum interference jitter. Based on the above discussion, the total number of arrivals of f_j can be computed as $\lfloor \frac{D_i+J_j^I}{T_j} \rfloor + 1$ (line 4, Alg. 2). Then we iteratively add the deterministic arrival time of each instance of f_j into the check-point list (line 5 - 7, Alg. 2). Each item of the check-point list CP contains two elements: a time instant Δt and the index of the flow who has a new arrival at Δt. At the end, the check-point list is resorted regarding an ascending order of Δt.

Once the check-point list is created, we can iteratively compute the response time distribution of f_i based on Eq. 1. We use $\mathcal{R}_i$ to represent the distribution of the response time of f_i, and we use $\mathcal{W}_i^l$ to denote the distribution of the cumulative workload considered during the analysis. The initial workload at time 0 consists of the transmission of f_i itself and the blocking caused by lower priority flits (line 4, Alg. 1). As discussed in Section IV-B, the upper-bound of the blocking delay for each packet can be computed by Eq. 6. Then we need to take into account the interference arrived at and after time 0 (line 5 - 17, Alg. 1). This is achieved by continuously adding interference at each deterministic check-point, since interference can only occur at the check-points.

Alg. 1 Calculation of response time distributions

1: **for all** $f_i \in S$, in a descending order of priorities **do**
2: $CP_i = CreateCP(f_i)$
3: $b_i = Eq.\ 6(f_i)$
4: $\mathcal{W}_i^l = \mathcal{C}_i + b_i$
5: **for all** p in CP **do**
6: **for all** w_k in $\mathcal{W}_i^l$ **do**
7: **if** $w_k < p.\Delta t$ **then**
8: $\mathcal{R}_i \leftarrow (w_k, Pr(W_i = w_k))$
9: $\mathcal{W}_i^l \rightarrow (w_k, Pr(W_i = w_k))$
10: **end if**
11: **end for**
12: **if** $\mathcal{W}_i^l = \varnothing$ **then**
13: break
14: **else**
15: $\mathcal{W}_i^l = \mathcal{W}_i^l \otimes \mathcal{C}_{p.index}$
16: **end if**
17: **end for**
18: **for all** w_k in $\mathcal{W}_i^l$ **do**
19: $\mathcal{R}_i \leftarrow (w_k, Pr(W_i = w_k))$
20: **end for**
21: $R_i^{max} = MIN(MAX(R_i), D_i)$
22: $J_i^I = R_i^{max} - C_i^{min}$
23: **end for**

At each check-point p, we first need to check if the cumulative workload can reach the time instant $p.\Delta t$. The NoC transmission uses a *work-conserving* scheduling policy, which means that once there are packets pending, the network will always transmit them. Therefore, for the cases where the workload cannot reach $p.\Delta t$ (i.e. $w_k < p.\Delta t$), we do not need to consider these cases in the following analysis, because all the workload (including f_i itself) has been completed before $p.\Delta t$. Accordingly, we remove these values of w_k ($w_k < p.\Delta t$) from the distribution of the cumulative workload $\mathcal{W}_i^l$ together with their probabilities (line 9, Alg. 1). At the same time, these values will be added into the response time distribution $\mathcal{R}_i$ (line 8, Alg. 1). An example is given in Fig. 4 showing how the distribution of $\mathcal{W}_i^l$ is split. In the example, W_i has 5 possible values (i.e. 1, 3, 5, 7 and 9). Assume that the next check-point is at time 4, then we can remove the values of 1 and 3 from $\mathcal{W}_i^l$. Because in these two cases, f_i already finishes its transmission before time 4, and it is impossible for f_i to get interference from packets arriving at or later than time 4. The remaining distribution of $\mathcal{W}_i^l$ is used during the further analysis.

Alg. 2 $CreateCP(f_i)$ /*create check-point list*/

1: $CP = [\ ]$ /*each item is formatted as $< \Delta t, index >$*/
2: **for all** $f_j \in S_i^D$ **do**
3: $CP \leftarrow < 0, j >$
4: $n = \lfloor \frac{D_i+J_j^I}{T_j} \rfloor + 1$
5: **for all** k in $[1, n-1]$ **do**
6: $CP \leftarrow < k \cdot T_j - J_j^I,\ j >$
7: **end for**
8: **end for**
9: sort CP regarding an ascending order of Δt
10: return CP

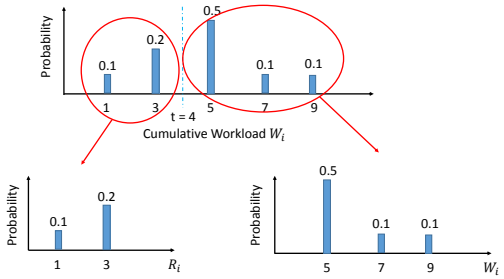

Fig. 4. An example showing how a distribution is split during the analysis.

The interference is added into the workload distribution by the convolution operation (line 15, Alg. 1). As the algorithm goes through more check-points, the algorithm will iteratively revise the upper tail of the workload distribution. On the other hand, as the cumulative probability in $\mathcal{W}_i^l$ decreases, the cumulative probability in $\mathcal{R}_i$ will increase accordingly. When the cumulative probability in $\mathcal{R}_i$ becomes 1, we get a complete distribution of the response times of f_i.

The analysis can be terminated by two conditions: (1) the remaining workload becomes 0 at a certain check-point (line 12-13, Alg. 1), which means that the upcoming interference cannot affect the response time of the packet under analysis at all; (2) the calculation has covered all the check-points, which means that all the possible higher priority packets arriving before the deadline D_i have been taken into account. When the first condition happens, the flow under analysis f_i will always meet its deadline. When the analysis terminates with the second condition, f_i is possible to miss its deadline.

As discussed earlier, while creating the check-point list of f_i, we need to take into account the interference jitter J_i^I of a higher priority flow f_j ($f_j \in S_i^D$). The calculation of J_i^I (Eq. 7) requires an upper-bound of the response time of f_j. Therefore,

we need to start the analysis with the flow who has the highest priority (line 1, Alg. 1).

The calculated response time distribution $\mathcal{R}_i$ can be utilized in different ways. If an application has requirements on the average response time, the expected value of the distribution can be used. The expected value of $\mathcal{R}_i$ can be computed as

$$E_i^R = \sum_{k=1}^{N_i} r_k \cdot Pr(R_i = r_k) \tag{8}$$

where E_i^R denotes the expected response time of f_i, and $\mathcal{R}_i$ contains N_i values.

From the computed distribution, the Deadline Miss Ratio (DMR) of each flow can also be calculated by

$$DMR_i = \sum_{\forall r_k > D_i} Pr(R_i = r_k) \tag{9}$$

where DMR_i represents the deadline miss ratio of f_i, and r_k denotes a possible value of R_i.

Moreover, using the Cumulative Distribution Function (CDF) of $\mathcal{R}_i$, we can obtain an upper-bound of the response time regarding a given probability. For example, assume that we get an upper-bound r_x from the CDF of $\mathcal{R}_i$ regarding a probability of 0.95. This result means that the response time of f_i has 0.95 probability to be lower than r_x. In other words, 95% of the packets of f_i have response times smaller than r_x. When the given probability is set to 1, the obtained upper-bound will be the same as the computed WCRT using the deterministic RTA. In general, stochastic RTA is more informative than the deterministic RTA.

E. Stochastic RTA of a NoC with Priority Sharing

Unfortunately, using a distinct priority for each flow can result in a large number of virtual-channels. Consequently, the required buffer size also increases, since each virtual-channel needs a certain amount of memory to buffer its packets. Due to the limited size of memory in most of the existing NoC routers, it is difficult to support a large amount of virtual-channels (e.g. the Intel Single Chip Cloud Computer only supports 8 virtual-channels). Therefore, it is necessary to employ certain mechanisms to handle a large amount of NoC flows with limited virtual channels. In this paper, we focus on the priority sharing policy [2]. Using this mechanism, multiple flows can have the same priority, which means that these flows can share the same virtual channel. In this case, the total number of virtual channels can be reduced.

Under the priority sharing policy, the flit-level preemptions only happen between different priority levels. In other words, one flow can only be preempted by a flow with higher priority. Within one priority level, the flows are scheduled based on a First-In-First-Out (FIFO) mechanism. In [2], the authors present a deterministic RTA for NoC flows with a priority sharing policy. The flows with the same priority level are analyzed based on the following theorem.

Theorem 2. *For a set of NoC flows with constrained deadlines ($D_i \leq T_i$ for any f_i), if they are schedulable while sharing the same priority, one flow can block any other flow at most once.*

Proof: This theorem can be proved by contradiction that if a flow can block another flow more than once, the condition of constrained deadline will be violated. More details have been explained in [2]. ■

Accordingly, the WCRT of a flow f_i happens when f_i is released together with all the flows which can cause direct interference to f_i (i.e. flows in S_i^D), and all the flows which share the same virtual-channel with f_i (i.e. flows in S_i^B) are released slightly earlier than f_i (i.e. the packets of all these flows are queued ahead of f_i).

Moreover, a flow f_p ($f_p \in S_i^B$) which is queued ahead of f_i may get interference from another flow f_q ($f_q \in S_p^D$). When the transmission of f_p is delayed by f_q, the response time of f_i will also be affected because f_p is blocking f_i at the same time. Therefore, even if $f_q \notin S_i^D$, f_q can still cause interference to f_i.

Similar to the analysis presented in Section IV-D, we need to first create a check-point list CP_i', where the above blocking and interference factors need to be taken into account. The creation of CP_i' is presented in Alg. 3. As discussed earlier, while analyzing the response time of f_i, we only need to consider one instance of each flow which shares the same virtual-channel with f_i (line 2-4, Alg. 3). All the flows which can cause direct interference to a flow f_p ($f_p \in S_i^B$) are approximately treated the same as the flows in S_i^D (line 5-11, Alg. 3). Once the new check-point list is generated, we can use Alg. 1 to compute the $\mathcal{R}_i$ of f_i.

Alg. 3 *CreateCP'*(f_i)

1: $CP = [\ ]$ /*each item is formatted as $< \Delta t, index >$*/
2: **for all** $f_p \in S_i^B$ **do**
3: $\quad CP \leftarrow < 0, p >$
4: **end for**
5: **for all** $f_j \in S_i^D \cup (\bigcup_{\forall f_p \in S_i^B} S_p^D)$ **do**
6: $\quad CP \leftarrow < 0, j >$
7: $\quad n = \lfloor \frac{D_i + J_j^I}{T_j} \rfloor + 1$
8: $\quad$ **for all** k in $[1, n-1]$ **do**
9: $\quad\quad CP \leftarrow < k \cdot T_j - J_j^I,\ j >$
10: $\quad$ **end for**
11: **end for**
12: sort CP regarding the ascending order of Δt
13: return CP

V. Evaluation

In this section, we present two sets of evaluation of our stochastic RTA. In the first set of evaluation, we focus on the pessimism included in the RTA. In the second set of evaluation, we aim to measure the processing time of the analysis in order to examine the scalability of the proposed approach.

A. Evaluation of Pessimism

As discussed in Section IV, in order to reduce the complexity of the analysis, approximations are utilized. In the first set of evaluation, we generate a number of experiments to investigate how much pessimism that can be involved. The pessimism is measured from the difference between the results

computed by the analysis and the samples obtained from simulations. The simulation result of each flow is observed from at least 1000 randomly collected samples. In this section, we use the *pessimism percentage* to show the evaluation results, which is calculated as

$$Pm_i = \frac{V_i^{RTA} - V_i^{SIM}}{V_i^{RTA}} \tag{10}$$

where Pm_i represents the pessimism percentage of the analysis result of f_i, V_i^{RTA} and V_i^{SIM} denote the analysis result and the simulation observation respectively.

We investigate the pessimism from three aspects: (1) the maximum response time of each flow (i.e. this will be the same as the result computed by a deterministic RTA); (2) the upper-bounds of response times with given probabilities of 0.98 and 0.95; (3) the average response time of each flow.

The on-chip network considered in our evaluation includes 4×4 cores, and it uses the X-Y routing algorithm. The flows are generated with random sources and destinations. The period of each flow is randomly generated from [100, 10000]. The total network utilization is selected from [0.2, 1.6]. Given the total network utilization, we use the UUnifast algorithm [23] to randomly generate the maximum utilization of each flow (i.e. U_i^{max}). The maximum transmission time of each flow C_i^{max} can then be computed by $T_i \cdot U_i^{max}$. The distribution of the transmission time of each flow contains 100 values where the values are randomly generated with the maximum value of C_i^{max}, and the corresponding occurrence probabilities are generated also using the UUnifast algorithm with the total probability of 1.

1) Evaluation of a NoC without Priority Sharing:

In this section, we present the results of the experiments where each flow uses a distinct priority.

First, we generate a number of experiments to investigate the relations between the total network utilization and the pessimism incurred in the analysis. In these experiments, the total number of flows is fixed to 100, and the total network utilization varies from set to set. The results are shown in Fig. 5. We separate the results into two groups. The first group contains flows which can get direct interference from at least one flow. As shown in Fig. 5-a, the pessimism percentages of the maximum response times are all around 15%, and the pessimism of the average response times are around 50%. As the total network utilization increases, there is no obvious trend of a change in the pessimism percentage. The second experiment group includes flows which do not experience any interference. As shown in Fig. 5-b, the pessimism percentages of both maximum and average response times are very low (i.e. less than 1.6%), which are much smaller compared to the first group. According to these results, we can observe that most of the pessimism is caused due to the approximations utilized in the calculation of interference.

Then, we generate another set of experiments to examine how the total number of flows in the network can affect the pessimism of the analysis. As shown in Fig. 6, when the total number of flows is small, the pessimism included in the results is low. As the number of flows increases, an increasing trend of the pessimism can be clearly observed. When the total number of flows increases from 10 to 300, the pessimism of the average response time goes up from 8% to 51%, and the pessimism of the maximum response time increases from 1% to 30%. The same trend can also be observed from the computed upper-bounds of response times with given probabilities of 0.95 and 0.98, where the pessimism increases from 8% to 50% and 6% to 46% respectively. As discussed earlier, the pessimism is mainly involved during the computation of the interference. While the number of flows goes up, a flow in the network has a higher chance to get interference from more flows, which results in more pessimism.

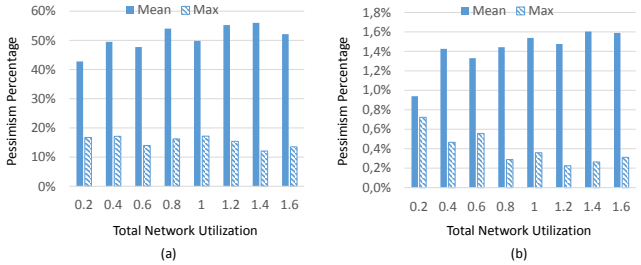

Fig. 5. The pessimism percentage regarding the total network utilization. Each value shown in this and the following figures represents the average result of 1000 flows.

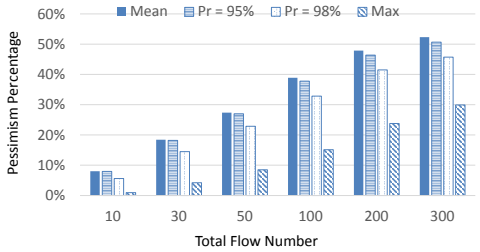

Fig. 6. The pessimism percentage regarding the total number of flows.

2) Evaluation of a NoC with Priority Sharing:

Similar experiments are also generated for the on-chip networks using a priority sharing policy. We generate two groups of experiments with different settings of virtual-channels.

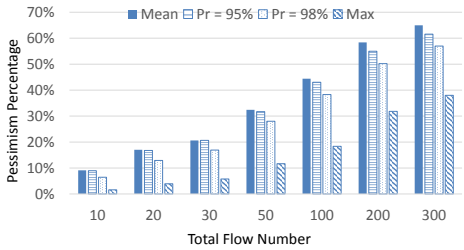

Fig. 7. The pessimism percentage regarding the number of flows, where the network contains 4 virtual-channels.

In the first group of experiments, the network is configured with 4 virtual-channels, and the results are presented in Fig. 7. As shown in the results, when the total number of flows increases from 10 to 300, the pessimism of the average response time goes up from 9% to 65%, and the pessimism of the maximum response time increases from 2% to 38%.

The pessimism included in the computed upper-bound with a probability of 0.95 is closer to the calculated average response time, which goes from 9% to 61%. The pessimism involved in the upper-bound with a given probability of 0.98 is slightly lower, which increases from 7% to 57%.

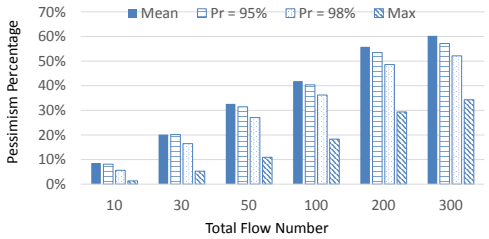

Fig. 8. The pessimism percentage regarding the number of flows, where the network contains 8 virtual-channels.

In the second group of experiments, the number of virtual-channels in the network is set to 8. As shown in Fig. 8, the pessimism percentage is slightly lower than the results with the configuration of 4 virtual-channels. Comparing to the results presented in Section V-A1, we can observe that the fewer virtual-channels a network uses, the more pessimism the analysis may involve. This is mainly because when we create the check-point list for f_i using Alg. 3, all the flows which can cause direct interference to f_p ($f_p \in S_i^B$) are considered as the direct interference flow of f_i. When the number of flows in S_i^B goes up, the number of flows in S_i^D will also increase. As a result, when f_i shares a virtual-channel with more flows, more flows need to be considered in S_i^D which results in more pessimism.

In general, we notice that the pessimism percentage regarding the maximum response time is much lower than the pessimism regarding the average response time, and the pessimism in the computed upper-bounds with probabilities of 0.98 and 0.95 is in between. As discussed earlier, the approximation during the calculation of interference can cause pessimism to all the above analysis results. However, comparing to the analysis of the maximum response time, the computation of the response time distribution includes additional pessimism. Because the calculation of the distribution uses the critical instant of each flow without taking into account the occurrence probability of the critical instant. In other words, while analyzing f_i, the analysis only uses the release patterns of all the other flows which can cause the worst-case situation to f_i. However, the occurrence probability of the critical instant is lower in reality. As a result, the peak of the calculated distribution will be shifted towards the upper tail (i.e. in the analysis results, larger values have higher probabilities than the reality, and lower values have lower probabilities than the reality). When we check the cumulative distribution functions (similar to Fig. 1), we can observe that the computed distribution of each flow is always greater than the distribution formed by the simulation results.

B. Evaluation of Processing Time

The time complexity of the stochastic RTA can be larger than the deterministic analysis due to the convolution operation. When we simply sum up two values in the deterministic RTA, we need to convolute two distributions in the stochastic RTA. As shown earlier in Example 1, a convolution of two variables involves much more computation than summing up two values. Apparently, the processing time of an analysis can directly affect its applicability. Therefore, in this section, we use measurements to investigate the scalability of the proposed stochastic RTA. The processing time is measured in true time from an analyzer developed using Python. The analyzer is executed in a system using Windows 7, equipped with an Intel Core(TM) i5-3320M CPU @2.60GHz and 8GB RAM.

The more values that a distribution has, the more representative it will be. However, using a large amount of values to represent each distribution will also cause a long processing time of the analysis. Therefore, we need to use re-sampling techniques to reduce the values included in each distribution while still providing acceptable analysis results. In this paper, we used the Uniform Spacing Re-sampling technique proposed in [24]. According to the experiments presented in [24], the Uniform Spacing Re-sampling technique requires fewer convolutions, and it can still provide representative results. In our experiments, we set the number of values in each distribution (i.e. N_i) to be 50 and 100.

In these experiments, networks with and without a priority sharing policy are both taken into account. The results are listed in Table I. As shown in the results of networks without priority sharing (i.e. the first 7 columns), when the total number of flows increases from 10 to 300, the maximum processing time using $N_i = 50$ increases from $1.96s$ to $29.077s$, while the maximum processing time with $N_i = 100$ goes from $5.17s$ up to $60.813s$. Together with results of networks with priority sharing (i.e. the last 6 columns), we can observe that setting $N_i = 100$ can cause much more processing time than the configuration of $N_i = 50$ (i.e. more than 2 times longer in any set of experiments). However, when we check the accuracy of these two settings, using $N_i = 100$ is just slightly better than the setting of $N_i = 50$. As shown in Fig. 9-a, when the total number of flows in the network is 50, the pessimism involved in the analysis using $N_i = 100$ is lower compared to using $N_i = 50$ with around 5% difference. However, when the total number of flows is 100 (Fig. 9-b), the pessimism included in the analyses with both setting is quite close (i.e. with around 2% difference). In summary, using $N_i = 50$ in the analysis can cause relatively lower processing time, while still provide acceptable accuracy.

On the other hand, we also notice that the analysis of networks using priority sharing obviously needs more processing time than the analysis of networks without priority sharing. Because given the same set of flows in the network, the S_i^D of a flow f_i under the configuration with priority sharing can include more flows compared to the setting without priority sharing.

In general, the processing time of the analysis can be acceptable in reality. When the total number of flows in the network is 300, the largest processing time observed from all these experiments is around $120s$.

	without Priority Sharing						with Priority Sharing (4 VCs)					
	N_i = 50			N_i = 100			N_i = 50			N_i = 100		
N_{total}	*MEAN*	*MAX*	*STD*	*MEAN*	*MAX*	*STD*	*MEAN*	*MAX*	*STD*	*MEAN*	*MAX*	*STD*
10	0.333	1.96	0.37	1.112	5.17	1.207	0.29	1.989	0.375	1.465	11.167	1.877
30	1.068	3.851	0.674	4.099	13.86	2.366	1.227	4.697	0.738	4.223	13.173	2.642
50	1.848	4.531	0.816	6.515	23.179	3.977	2.526	8.074	1.274	7.751	23.445	4.617
100	4.336	10.492	1.447	11.645	37.795	5.277	7.986	24.13	3.354	18.658	50.367	8.237
200	9.044	16.184	2.311	21.516	42.92	7.629	17.688	39.927	5.773	38.806	85.484	13.782
300	18.738	29.077	3.361	35.997	60.813	10.71	30.674	47.407	6.614	67.158	120.08	17.861

TABLE I

THE RESULTS SHOWING THE PROCESSING TIMES (IN *s*) OF THE ANALYSIS. NOTE THAT THE RESULTS CAN INCLUDE SLIGHT INACCURACY DUE TO THE OPERATING SYSTEM ENVIRONMENT.

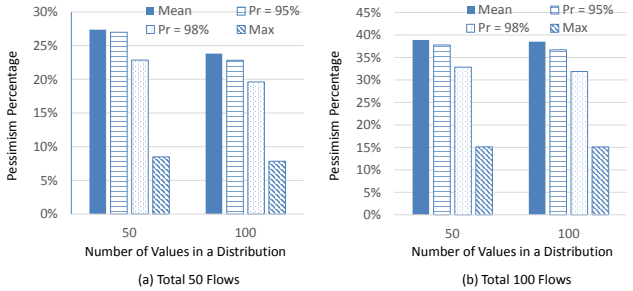

Fig. 9. The pessimism percentage regarding the number of values in each distribution.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present a stochastic response time analysis for wormhole-switched on-chip networks. The proposed analysis calculates a probability distribution of response times of each NoC flow as results. The computed distribution can be used to obtain upper-bounds regarding different probabilities, average response times, or deadline miss ratios. Therefore, this approach can provide more informative results compared to the existing deterministic RTA. A number of simulation-based evaluations are also implemented in order to investigate the pessimism involved in the analysis. The evaluation results show that the pessimism increases as the total number of flows in the network goes up. When the total number of flows is 300 in a network without priority sharing, the pessimism of the computed maximum response time is around 30%, and the pessimism in the estimated average response time is around 50%. When we decrease the number of virtual-channels in the network (i.e. using priority sharing), the involved pessimism increases. The processing time of the analysis during the experiments is also measured in order to examine the scalability of the proposed approach. According to the results, the processing time of the analysis can be acceptable in reality.

One of the most important future works is to reduce the pessimism involved in the analysis, since pessimism can lead to waste of system resources and it limits the applicability of the analysis. On the other hand, in this paper, we assume that the packet size is the only probabilistic parameter of each flow. We will extend the analysis to handle NoC flows with multiple stochastic parameters in order to support more general applications. Moreover, the proposed analysis assumes that the priority of each flow is already given. It is necessary to investigate how to assign priorities of flows in the context of NoCs with probabilistic real-time constraints.

REFERENCES

[1] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *NOCS*, 2008.

[2] ——, "Real-time communication analysis with a priority share policy in on-chip networks," in *ECRTS*, 2009.

[3] D. Dasari, B. Nikoli'c, V. N'elis, and S. M. Petters, "Noc contention analysis using a branch-and-prune algorithm," *ACM Trans. Embed. Comput. Syst.*, 2014.

[4] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Stochastic analysis of can-based real-time automotive systems," *Industrial Informatics, IEEE Transactions on*, 2009.

[5] ——, "Using statistical methods to compute the probability distribution of message response time in controller area network," *Industrial Informatics, IEEE Transactions on*, 2010.

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

[7] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, 2002.

[8] N. K. Kavaldjiev and G. J. M. Smit, "A survey of efficient on-chip communications for soc," in *4th PROGRESS Symposium on Embedded Systems*, 2003.

[9] J. Diemer and R. Ernst, "Back suction: Service guarantees for latency-sensitive on-chip networks," in *NOCS*, 2010.

[10] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *RTCSA*, 2008.

[11] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," *Design Test of Computers, IEEE*, 2005.

[12] T. Ferrandiz, F. Frances, and C. Fraboul, "A sensitivity analysis of two worst-case delay computation methods for spacewire networks," in *ECRTS*, 2012.

[13] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001.

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

[15] M. K. Gardner and J. W. Liu, "Analyzing stochastic fixed-priority real-time systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, 1999.

[16] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *RTSS*, 2002.

[17] D. Maxim and L. Cucu-Grosjean, "Response time analysis for fixed-priority tasks with multiple probabilistic parameters," in *RTSS*, 2013.

[18] J. M. López, J. L. Díaz, J. Entrialgo, and D. García, "Stochastic analysis of real-time systems under preemptive priority-driven scheduling," *Real-Time Systems*, 2008.

[19] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, 1986.

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

[21] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, 2007.

[22] C. Fallin, X. Yu, G. Nazario, and O. Mutlu, "A high-performance hierarchical ring on-chip interconnect with low-cost routers," *Computer Architecture Lab, Carnegie Mellon Univ, Tech. Rep*, 2011.

[23] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, 2005.

[24] D. Maxim, M. Houston, L. Santinelli, G. Bernat, R. I. Davis, and L. Cucu-Grosjean, "Re-sampling for statistical timing analysis of real-time systems," in *RTNS*, 2012.