SLA: A Stage-level Latency Analysis for Real-time Communication in a Pipelined Resource Model

Hany Kashif, Sina Gholamian, and Hiren Patel

Abstract—We present a communication analysis for hard real-time systems interconnects. The objective is to provide tight estimates on the worst-case communication latency between communicating processing elements that use a priority-aware communication medium for data transmission. The communication model consists of communication tasks transmitting data across a series of pipelined resources. The analysis incorporates interferences caused by multiple communication tasks requesting the pipelined resources, and it captures parallel transmission of data between multiple pipeline stages. We call this analysis a stage-level analysis. We evaluate the proposed analysis through simulation of synthetic benchmarks, and we apply the analysis to an instantiation of a platform proposed by Shi and Burns. Our experiments confirm that stage-level analysis provides tight upper-bounds when compared to previous work and improves schedulability by 34%.

Index Terms—Real-Time Communication, and Worst-Case Response Time Analysis.

1 Introduction

Distributed hard real-time systems may consist of periodic and sporadic tasks deployed on multiple processors communicating with each other through a communication interconnect. For the correct operation of the real-time system, a guarantee must be provided that the entire system is schedulable. This is done by ensuring that the temporal requirements of all tasks are met, which requires an estimate on the worst-case execution time (WCET) of tasks and the worst-case communication latency (WCL) between the tasks. While there is significant work on estimating WCET of tasks, the WCL between tasks is often side-stepped. Providing guarantees on multiprocessor interconnects used in general purpose computing is difficult. This is because conventional communication interconnects focus on improving the average-case performance via dynamic routing algorithms, and flow control policies. These, however, are a hindrance to accurately predicting the WCL. Consequently, researchers proposed customized interconnects specifically for its use with hard real-time systems [1], [2].

The two common approaches used in customizing interconnects use resource reservation or run-time arbitration. Resource reservation allocates resources before the start of communication to ensure that there is no contention for resources between any two tasks. An example of a resource reservation approach is time-division multiplexing (TDM) [3], [1], [4]. This approach statically allocates slots to communication tasks such that no other task can use that slot other than the assigned one. In the event that there is no data to be transmitted in that slot, the slot remains empty; thereby, not allowing other tasks to use it resulting in under-utilization of communication resources. For low latency communication, a larger proportion of slots are assigned; thus, causing low latency communication to be intertwined with the bandwidth when using TDM. Finally, TDM does not cleanly support sporadic tasks. This is because sporadic tasks are triggered by external events and, hence, it is unknown at design time when exactly they start.

Run-time arbitration, on the other hand, arbitrates access to communication resources at run-time. Hence, contention is expected, and the analysis accounts for these contentions to produce the WCL. One such communication architecture was proposed by Shi and Burns [2], [5] that supports wormhole switching with priority-based arbiters that allow higher priority communication to preempt lower ones. Moreover, this approach overcomes the tight coupling of latency and bandwidth suffered by TDM and TDM-like approaches, and it allows for a variety of communication task types with its use of priorities. Shi and Burns also present a WCL analysis, which we call flow-level analysis (FLA) [2], [5] that determines the WCLs between communicating tasks. Their analysis includes direct and indirect interferences from other communications. This analysis is central in determining the schedulability of the communication tasks. However, FLA assumes a model where the tasks are indivisible units of communication. As a consequence, FLA does not incorporate the effects of pipelining and parallel transmission of data in

• *H. Kashif, S. Gholamian, and H. Patel are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, N2L 3G1.*
E-mails: hkashif@uwaterloo.ca, sgholamian@uwaterloo.ca, hdpatel@uwaterloo.ca

0000–0000/00/$00.00 © 2014 IEEE Published by the IEEE Computer Society

its communication model. This restriction in the model results in higher upper-bounds on the communication latencies.

In this article, we address this issue by proposing a pipelined communication resource model for analyzing the WCLs for hard real-time systems. This model supports pipelined and parallel transmission of data over communication resources with fixed priority preemption. We also present an associated stage-level analysis (SLA) that uses the pipelined communication resource model to produce tight WCL estimates. The model supports communication tasks that are either periodic or sporadic. This analysis is suitable for interconnects that use run-time arbitration such as that proposed by Shi and Burns. However, note that we do not present an interconnect architecture in this work, but a model upon which we can compute the WCL. To evaluate the strength of the model and its analysis, we create instances of the interconnects proposed by Shi and Burns, and show that SLA provides tighter WCLs than FLA, both through analysis, and simulation experiments. Our simulation experiments show that SLA improves the number of schedulable task sets by 34%.

To further show the applicability of SLA, we use the presented analysis in the following works [6], [7], [8]. The first work addresses the problem of selecting the routes for communication [6] so as to increase schedulability of the entire deployment. The second work [7] develops a worst-case response time analysis for multiple applications deployed on a multiple processor system with an interconnect analyzable using SLA. The work in [7] uses SLA but focuses on computing the worst-case response times of multiple applications that have dependent computation and communication tasks. The last work uses SLA to discover the buffer requirements of the routers in the interconnect [8].

1.1 Main contribution

We propose a stage-level WCL analysis. The main contributions of this work are as follows.

1) A pipelined communication resource model, and its associated communication task model supporting periodic and sporadic tasks.
2) A WCL analysis under the condition that task deadlines are less than their corresponding periods (no self-blocking).
3) A generalized WCL analysis that relaxes the deadline assumption to include self-blocking from the task under analysis.
4) An analytical comparison with FLA to prove the merits of SLA.
5) Extensive experimental evaluation of the analysis to determine schedulability gains compared to FLA.

1.2 Organization

This paper is organized as follows: Section 2 discusses related work. Sections 3 and 4 present the resource model and the communication task model, respectively. Sections 5.1 and 5.3 formally define direct and indirect interferences, respectively. Section 5.2 presents the WCL analysis using only direct interferences and under the assumption deadlines are less than their corresponding period. Section 5.4 extends the latency analysis to include both direct and indirect interferences. Section 5.5 provides an example that illustrates SLA. Section 5.6 proves that SLA has tighter WCL bounds compared to FLA. Section 5.7 derives a WCL bound after relaxing the deadline assumption. Section 6 presents and discusses the experimentation results. Finally, Section 7 concludes the paper.

2 Related Work

There exists other techniques to compute end-to-end worst-case delays on networks. These include network calculus [9], an extension of network calculus for real-time systems called real-time network calculus [10], holistic analysis [11], [12], [13], delay calculus [14], [15], and flow-level analysis [2], [5]. We discuss these related works in the context of the proposed approach.

Network calculus [9] is a deterministic queuing theory that uses max/min-plus algebra to determine performance bounds on load and service of a network. It is based on analyzing dataflows (that can be computation or communication) to generate cumulative functions of events entering a node and departing a node. Real-time calculus [10] specializes network calculus for real-time embedded systems by characterizing dataflows as interval bound functions, and incorporating methods to model schedulers [16]. Traditional worst-case bound methods in network calculus assume blind multiplexing. This requires that no assumptions are made for the arbitration of multiple flows traversing a server. It is known that tighter upper-bounds can be computed by providing insights of the multiplexings [17]. Real-time calculus enables this with models of its arbitration schedulers. Furthermore, real-time calculus uses both upper and lower bound curves on arrival and service to compute worst-case delays, which contribute to tighter worst-case delays estimates. However, a difficulty shared by these approaches is that of analyzing flows with certain traversal patterns such as those that are in nested and non-nested tandem [17]. A nested tandem situation is one where the path flows taken are either nested within the flow under analysis or disjoint. This difficulty arises because these approaches convolve arrival/service curves of each node; hence, the pipelined behaviour of the network is not considered. This results in double-counting of events (repeatedly accounting for events that have already caused interferences at an earlier node) in the dataflow yielding pessimistic worst-case bounds. Our approach differs compared to these in that the analysis is not as general as network or real-time calculus. This allows us to incorporate details of the arbitration scheme, and the pipeline behaviour of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

network. This enables us to remove events that result in double-counting of interferences does by understanding that once a flow has interfered, then its re-interference does not extend the worst-case latency.

Holistic analysis [11], [12], [13] introduces a worst-case response time analysis [18], [19] for transactional task models by considering task offsets. These tasks offsets are combined with jitters to characterize the arrival pattern of tasks at each node. The offset represents the earliest a task can be released, and the jitter represents the worst-case. Using the offset and jitter of a task arriving on a particular node, holistic analysis computes the offset and jitter of the task leaving that node. This becomes the arrival pattern for tasks for the next stage. Holistic analysis iteratively computes these patterns across all nodes resulting in the end-to-end worst-case latency of a given task. The primary difference between holistic analysis and the proposed approach is that holistic analysis does not take into account relationships between different resources. Thus, it does not leverage the pipelined transmission of data, which the proposed approach does in order to deliver a tighter upper-bound.

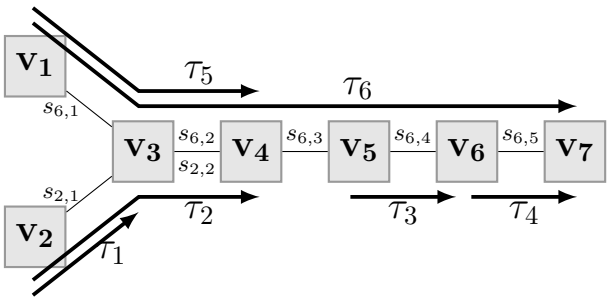
Delay calculus [14], [15] proposes a method to compute end-to-end worst-case delays experienced by jobs executed over multiple stages. Unlike holistic analysis, delay calculus does incorporate pipelined behaviour in its analysis. However, the preemption model is significantly different than the one of the proposed approach. A job in delay calculus may be preempted midway during execution, and resumed later once the higher priority job completes. This does not apply to SLA where a flit is either successfully routed to the output port or it is preempted by a higher priority flit. Hence, a flit is either preempted in its entirety or not. We find that the model adopted by SLA respects router models where midway transmissions are usually not preempted. Additionally, delay calculus only partially considers tandem situations. We find that SLA employs the preemption model that adheres to convention in network routers, and the analysis does address tandem situations.

Shi and Burns [2], [5] propose what we term the flow-level analysis, which uses response-time analysis techniques to deliver worst-case latency bounds. This approach supports multiple priorities, and their router uses the same preemption model adopted by SLA. FLA considers flows as indivisible transmissions across multiple nodes. As a result, it does not incorporate the effects of pipelined and parallel transmission of data. This yields grossly pessimistic worst-case latency bounds. Furthermore, under certain nested tandem situations the bounds cannot be computed; however, in reality these are computable, which is done by SLA. SLA allows for pipelined and parallel transmission, and it further tightens the worst-case latencies by removing double-counting of interferences.

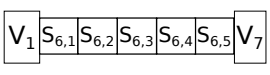
3 Resource Model

The resource model consists of pipelined communication resources that connect the computation resources. The computation tasks execute on the computation resources and communicate using communication tasks that execute on the communication resources. The communication tasks that execute over the communication resources support fixed priority preemption. A communication task can execute over multiple communication resources or stages. Figure 1a shows a deployment of communication tasks to communication resources. Each node represents a computation resource and each edge represents a communication resource. Figure 1b shows the resource model for communication task τ_6. The communication task τ_6, used for the communication from the computation task running on node V_1 to computation task running on node V_7, executes on five stages. These stages are pipelined and have the following characteristics:

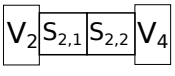
1) A path for the communication between two computation tasks is a sequence of one or more stages. A communication task accesses the stages of its path in an orderly fashion.
2) Data is transmitted on stages in parallel, i.e., when data moves from an earlier to a later stage, new data can be transmitted on the earlier stage in parallel. In Figure 1b, after a data unit is transmitted on stage $s_{6,1}$, it will move to stage $s_{6,2}$ and a new data unit can be transmitted on stage $s_{6,1}1$.
3) If a data unit is transmitted at time t on one stage, then it will be ready for transmission at time $t + R_{s_{i,l}}$ on the next stage (where $R_{s_{i,l}}$ is a delay associated with accessing a new stage). This data unit will be transmitted on the next stage at time $t + R_{s_{i,l}}$ unless it is preempted by a higher priority data unit.



(a) Deployment of communication tasks to resources.



(b) Model for τ_6.



(c) Model for τ_2.

Task	L	T	D	J
τ_1	2	8	8	0
τ_2	2	8	8	0
τ_3	2	8	8	0
τ_4	2	8	8	0
τ_5	2	8	8	0
τ_6	9	50	50	0

(d) Communication task data.

Fig. 1: Motivating example.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

If multiple data units are ready for transmission on a stage, then the higher priority one will be transmitted on the stage. Figure 1a shows that tasks τ_6 and τ_2 share the same second stage $s_{6,2} = s_{2,2}$. If a data unit from τ_6 and another from τ_2 attempt to transmit on $s_{6,2}$ (or $s_{2,2,}$)at time t, the higher priority data unit will be transmitted on $s_{6,2}$. The lower priority data unit will only transmit when there are no higher priority data units pending transmission.

Since communication tasks support fixed priority preemption, then a higher priority task can preempt the execution of a lower priority task on any stage. Hence, in the proposed model, buffers are required at the computation resources and between the communication resources. We assume in this work that there is enough buffer space to store data. Note as well, that we can derive an upper-bound on the buffer space requirements using the stage-level analysis. Note, also, that since the tasks support fixed priority preemption, lower priority tasks might starve for communication resources. The analysis, presented later in this article, finds the WCL for each communication task and, hence, can detect if starvation occurs.

4 Communication Task Model

We present the definitions, terminology and the model necessary for describing the stage-level analysis. In our analysis, we assume the assignment of distinct priorities to communication tasks. We assume that a set of communication tasks Γ is deployed on the communication resources. We also assume that communication tasks have fixed paths that are determined offline.

Definition 1. *The set of communication tasks* $\Gamma := \{\tau_i : \forall i \in [1, n]\}$ *has* n *communication tasks, where a communication task* τ_i *is a 5-tuple* $\langle P_i, T_i, D_i, J_i^R, L_i \rangle$*. This describes a communication task* τ_i *with priority* P_i*, period* T_i *between successive job transmissions, real-time deadline* D_i*, release jitter* J_i^R*, and the basic stage latency* L_i*.*

A communication task transmits data from a source computation resource to a destination. A communication task is schedulable if its WCL R_i is less than the deadline D_i. The release jitter J_i^R is the worst-case delay in a job's release time. Communication tasks can be periodic tasks with a period T_i or sporadic tasks with a minimum interarrival time T_i between jobs. The basic stage latency, L_i, is the WCL of a job of the task on one stage when it does not suffer interferences from any other tasks on that stage.

The path that the communication task τ_i traverses is a sequence of stages denoting multiple communication resources that it crosses to reach from the source computation resource to the destination.

Definition 2. *A path* δ_i *for communication task* τ_i *is a sequence of stages* $(s_{i,1}, \ldots, s_{i,|\delta_i|})$*.*

Definition 3. *A subpath* $\sigma_i(s_{i,l})$ *for communication task* τ_i *is a sequence of stages* $(s_{i,1}, \ldots, s_{i,l})$ *such that* $\sigma_i(s_{i,l})$ *has the same first stage* s_1 *as path* δ_i*, and* $l < |\delta_i|$ *with* $s_{i,l}$ *being the last stage of the subpath.*

We use $|\delta_i|$ to denote the number of stages in path δ_i. The basic latency of a communication task τ_i along its path δ_i is its execution time on all stages along its path without suffering any interference. The term $R_{s_{i,l}}$ is the delay experienced by a data unit as it moves from stage $s_{i,l-1}$ to stage $s_{i,l}$. Since the stages are pipelined and tasks experience a delay $R_{s_{i,l}}$ when moving from one stage to another, we can compute the basic latency of a communication task as $C_i = L_i + \sum_{l=2\ldots|\delta_i|} R_{s_{i,l}}$. Note that a higher priority data unit, moving from one stage to another, can be blocked by a lower priority data unit that has already started transmission. This blocking time (at each stage) is at most the latency of transmitting one data unit. This time is constant at each stage. For clarity, we drop the blocking time from our analysis and derivations given that it is a constant that can be added to the WCLs presented in this work.

We use the notation $s \in \delta_i$ to denote that a stage s exists on the path δ_i. We also use $\delta_i \cap \delta_j$ to denote the set of stages that exist on both paths δ_i and δ_j. For $s, s' \in \delta_i$, we use tick (s') to denote a stage that precedes another stage s in the path δ_i.

We present an illustrative example in Figure 1a to familiarize the reader with the terminology. Figure 1a maps 6 communication tasks to communication resources: $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$ with the priorities $P_1 > P_2 > P_3 > P_4 > P_5 > P_6$. Edges between nodes represent communication resources. Note that the choice of our paths in Figure 1a are selected solely to illustrate, and explain the stage-level analysis. Table 1d presents the basic stage latency, the period (or minimum interarrival time), the deadline, and the jitter for each of the tasks in Figure 1a. For simplicity, the tasks have zero jitter and the task deadlines are equal to their periods.

We use Figure 2 to show the transmission of one job of τ_6 on each of the stages in the path δ_6 for the example configuration in Figure 1a. We assume a per stage delay $R_{s_{i,l}} = 1$. For example, since τ_5 has a higher priority than τ_6, jobs of τ_5 will preempt the job of τ_6. White spaces on stage (v_4, v_5) represent gaps caused by interfering tasks on the predecessor stage (v_3, v_4). These tasks no longer interfere with τ_6 on stage (v_4, v_5), but the data units of τ_6 remain separated by the gaps shown due to interferences on previous stages. Note that in our derivation of an upper-bound for the latency, we assume that these gaps are part of the latency of task τ_6 as we show in more detail in Section 5.

5 Stage-level Analysis (SLA)

Stage-level analysis (SLA) incorporates direct and indirect interferences with other communication tasks. We describe this analysis with the use of Figure 2 as a running example to provide the intuition behind the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

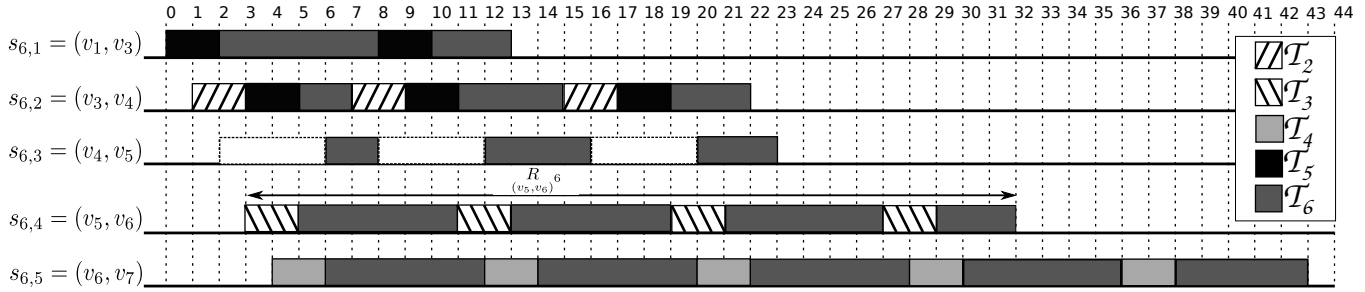


Fig. 2: Timeline of task τ_6 in Figure 1a.

mathematical formulation. We first present the WCL with direct interference of tasks on a particular stage. We use this to compute the WCL of the communication tasks' entire path, which we later augment with indirect interferences. Henceforth, we use the term "task" to refer to a communication task.

5.1 Direct Interference

Direct interference occurs when a higher priority task τ_j preempts a lower priority task τ_i on a shared stage. We formalize direct interference at the stage-level in Definition 4, and the set of tasks resulting in direct interferences on a stage in Definition 5.

Definition 4. *A task τ_i suffers direct interference from task τ_j on a stage s if and only if $s \in \delta_i \cap \delta_j$, and $P_i < P_j$.*

Definition 5. *The set of tasks directly interfering with task τ_i on a stage s is $\mathbb{S}_{s}^{D}{}_{i} = \{\tau_j \mid \forall \tau_j \in \Gamma, s \in \delta_i \cap \delta_j \text{ and } P_i < P_j\}$.*

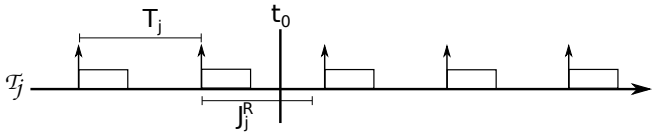
From Figure 1a, we observe that τ_6 has direct interference on its second stage $s_{6,2} = (v_3, v_4)$ from tasks τ_2 and τ_5 such that $\mathbb{S}_{(v_3,v_4)}^{D}{}_{6} = \{\tau_2, \tau_5\}$.

5.2 Worst-case Latency with Direct Interference

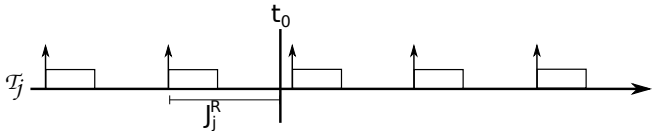
We derive the WCL for the special case of $D_i \leq (T_i - J_i^R)$. This case prevents interference of jobs of the task under analysis (τ_i) with another job of the same task. This is commonly referred to as self-blocking. Later, we relax this restriction. We define the worst-case contribution of a higher priority task to the latency of a task under analysis. We then derive the WCL for the first stage on the path of the task under analysis, followed by deriving the WCL for any arbitrary stage on the path. Figure 3a shows a sample activation pattern for the jobs of a higher priority task τ_j directly interfering with a task under analysis τ_i. The time instant t_0 represents the release of the job under analysis of task τ_i.

Lemma 1. *The worst-case contribution from a higher priority task τ_j to the latency of a job under analysis of task τ_i on a stage $s \in \delta_i$ is achieved when the following rules are applied:*

1) *Jobs of τ_j that are activated before t_0 and have enough jitter to be released at t_0 are released at t_0.*



(a) Example pattern.



(b) Modified pattern.

Fig. 3: Activation pattern of task τ_j.

2) *Jobs of τ_j that are activated after t_0 are released immediately with no jitter.*
3) *The release of one of the jobs of τ_j coincides with t_0 after experiencing maximum jitter.*

Proof: These three rules lead to a larger WCL for the job under analysis. The first rule is concerned with jobs of τ_j that are activated before t_0, but can be released at or after t_0. If these jobs are released before t_0, they will either not interfere with the job under analysis or will cause less interference compared to being released at t_0. Also if these jobs are released after t_0, they might not contribute to the WCL of the job under analysis. Hence, releasing the jobs at t_0 leads to maximum interference.

The second rule states that jobs activated after t_0 should be immediately released. Since these jobs are activated after t_0, then they might interfere with the job under analysis. The earlier the jobs are released, the higher the chance they will cause interference. Hence, the immediate release of these jobs contributes most to the WCL of the job under analysis.

The third rule states that the contribution of τ_j to the latency of the job under analysis is worst when the release of one of its jobs coincides with t_0 after suffering maximum jitter. Consider, for instance, the activation pattern in Figure 3a. Assume (without loss of generality), that according to the first two rules, only one job of τ_j has enough jitter to be released at t_0 (first job to the left of t_0) and two jobs activated after t_0 contribute to the latency of the job under analysis (first two jobs to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

the right of t_0). Moving the activations of the jobs of τ_j earlier in time (as shown in Figure 3b) until the first job (released at t_0) has maximum jitter while still being released at t_0, contributes most to the latency of the job under analysis. The reason is that by moving activations earlier in time, jobs of τ_j that were activated after t_0 (the last job to the right) and could not interfere with the job under analysis might be able to cause interference. □

Lemma 2. *The WCL $\underset{s}{R}_i$ of a task τ_i suffering direct interferences from other tasks in $\underset{s}{\mathbb{S}}_i^D$ on the first stage $s = s_{i,1} \in \delta_i$ under the condition $D_i \leq (T_i - J_i^R)$ is given by the following:*

$$\underset{s}{R}_i = \underset{s}{I}_i + L_i$$

where

$$\underset{s}{I}_i = \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s}{R}_i + J_j^R}{T_j} \right\rceil * L_j$$

Proof: Given $D_i \leq (T_i - J_i^R)$, a job under analysis of task τ_i does not suffer interference from other jobs of the same task. Assume $\underset{s}{R}_i$ is the upper-bound on the WCL of a job under analysis of task τ_i. Consider a higher priority task $\tau_j \in \underset{s}{\mathbb{S}}_i^D$ that directly interferes with task τ_i. From Lemma 1, the interval of time during which τ_j can interfere with the job under analysis is the WCL of the job plus the maximum release jitter of τ_j, i.e., $\underset{s}{R}_i + J_j^R$. The maximum number of jobs of task τ_j that interfere with τ_i in that interval of time is thus equal to $\left\lceil \frac{\underset{s}{R}_i + J_j^R}{T_j} \right\rceil$. The latency contribution of τ_j to the WCL of τ_i is equal to the maximum number of interfering jobs multiplied by the basic stage latency of τ_j, i.e., $\left\lceil \frac{\underset{s}{R}_i + J_j^R}{T_j} \right\rceil * L_j$. Therefore, the WCL of τ_i on stage $s_{i,1}$ is the summation of the interference from all higher priority tasks in the set $\underset{s}{\mathbb{S}}_i^D$ plus the basic stage latency of τ_i, i.e., $\underset{s}{I}_i + L_i$. This results in a recurrence relation that can be solved iteratively to find $\underset{s}{R}_i$. □

Under the assumption that $D_i \leq (T_i - J_i^R)$ and using only direct interference, so far we have derived the WCL of τ_i on the first stage of its path δ_i. Next, we derive the WCL of τ_i on any arbitrary stage s along its path δ_i. The terms $\underset{s'}{R}_i$ and $\underset{s'}{I}_i$ represent the WCL and the worst-case interference of task τ_i on a stage s' preceding a stage s on the path δ_i, respectively.

Lemma 3. *The WCL $\underset{s}{R}_i$ is monotonically increasing with respect to the stages of the path δ_i.*

Proof: $\underset{s'}{R}_i$ is the worst-case time interval between the first and last data units of the job of τ_i on stage s'. If no new interferences exist on stage s (compared to those on stage s'), then the WCL $\underset{s}{R}_i$ is equal to $\underset{s'}{R}_i$. The reason is that any data unit that was transmitted at a time t on stage s' will attempt transmission at time $t + R_{s_{i,l}}$ on stage s with the same ordering of data units as on s'. Assume a higher priority data unit was transmitted at time t on stage s' followed by a data unit of τ_i at time $t+1$. If the higher priority data unit does not exist on stage s, then the slot at time $t + R_{s_{i,l}}$ will be empty. This will not cause the τ_i data unit to transmit any earlier on stage s as it will attempt transmission at time $t+1+R_{s_{i,l}}$ leaving a gap at the time $t + R_{s_{i,l}}$. Therefore, the WCL on stage s is greater than or equal that on stage s', i.e., $\underset{s}{R}_i \geq \underset{s'}{R}_i$. □

Lemma 4. *The WCL $\underset{s}{R}_i$ of a task τ_i suffering direct interferences from other tasks in $\underset{s}{\mathbb{S}}_i^D$ on stage $s \in \delta_i$ is given by:*

$$\underset{s}{R}_i = \underset{s}{I}_i + L_i$$

where

$$\underset{s}{I}_i = \underset{s'}{I}_i + \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s}{R}_i + J_j^R}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D \cap \underset{s'}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s'}{R}_i + J_j^R}{T_j} \right\rceil * L_j$$

Proof: **Base Case:** Note that in the interference term $\underset{s}{I}_i$, and for the first stage $s = s_{i,1}$, the stage s' does not exist. Hence, for the first stage, $\underset{s'}{I}_i = 0$. Also the set $\underset{s}{\mathbb{S}}_i^D \cap \underset{s'}{\mathbb{S}}_i^D = \emptyset$, which makes the last term $\sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D \cap \underset{s'}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s'}{R}_i + J_j^R}{T_j} \right\rceil * L_j = 0$. Therefore, for the first stage, the interference term becomes the one derived from Lemma 2. This provides the base case for our proof. We prove Lemma 4 by induction. Assume Lemma 4 holds for an arbitrary stage $s' \in \delta_i$. We now prove Lemma 4 for the succeeding stage $s \in \delta_i$.

Consider three higher priority tasks: τ_j, τ_k, and τ_l that interfere with task τ_i. Assume that τ_j interferes with τ_i only on stage s', τ_l interferes on stage s only, and that τ_k causes interference on both stages s' and s. For simplicity (and without loss of generality), assume that these are the only interfering tasks with τ_i on the stages s' and s such that $\underset{s'}{\mathbb{S}}_i^D = \{\tau_j, \tau_k\}$ and $\underset{s}{\mathbb{S}}_i^D = \{\tau_k, \tau_l\}$.

In Lemma 3, we illustrated that any higher priority jobs that caused interference to τ_i on s' cannot cause any more interference on stage s. If any new interference occurs on stage s, then it will delay the data units arriving from s' by an interval of time equivalent, in the worst-case, to the interference caused by the new jobs. Hence, a conservative approach to computing the latency of τ_i on stage s would be by adding the interferences from τ_k and τ_l on stage s to the latency computed on stage s':

$$\underset{s}{R}_i = \underset{s'}{R}_i + \left\lceil \frac{\underset{s}{R}_i + J_k^R}{T_k} \right\rceil * L_k + \left\lceil \frac{\underset{s}{R}_i + J_l^R}{T_l} \right\rceil * L_l \quad (1)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

We, however, can further tighten the upper-bound on the latency of τ_i on stage s. Note that in the Equation 1 (conservative approach), we computed $\underset{s}{R_i}$ to include interference from jobs of τ_k, although, some of these jobs have already been accounted for in $\underset{s'}{R_i}$. Since higher priority jobs that caused interference to τ_i on s' cannot cause any more interference on stage s, then we can achieve a tighter latency bound by ensuring that, in the computation of $\underset{s}{R_i}$, we only add interferences that were not accounted for in $\underset{s'}{R_i}$. The WCL on stage s', $\underset{s'}{R_i}$, includes interferences in the set $\underset{s'}{\mathbb{S}}_i^D = \{\tau_j, \tau_k\}$. The WCL on stage s, $\underset{s}{R_i}$, should include interferences in the set $\underset{s}{\mathbb{S}}_i^D = \{\tau_k, \tau_l\}$. Note, however, that τ_k is a common task, i.e., $\underset{s'}{\mathbb{S}}_i^D \cap \underset{s}{\mathbb{S}}_i^D = \{\tau_k\}$ so we want to only add interference from τ_l and any new jobs from τ_k that did not exist on s'. Also note that, from Lemma 3, the WCL $\underset{s}{R_i}$ is monotonically increasing with respect to the stages on path δ_i. Hence, the number of jobs of τ_k causing interference on stage s can only be greater than or equal to those on stage s'. The number of jobs of τ_k that caused interference on stage s' is $\left\lceil \frac{\underset{s'}{R_i}+J_k^R}{T_k} \right\rceil$. And the number of τ_k jobs that causes interference on stage s is $\left\lceil \frac{\underset{s}{R_i}+J_k^R}{T_k} \right\rceil$. Therefore, a tighter bound can be achieved by subtracting the latency of the jobs of τ_k that already existed on stage s' and will definitely exist in the jobs interfering on stage s:

$$\underset{s}{R_i} = \underset{s'}{R_i} + \left\lceil \frac{\underset{s}{R_i} + J_k^R}{T_k} \right\rceil * L_k + \left\lceil \frac{\underset{s}{R_i} + J_l^R}{T_l} \right\rceil * L_l - \left\lceil \frac{\underset{s'}{R_i} + J_k^R}{T_k} \right\rceil * L_k$$

Since $\underset{s}{R_i} = \underset{s}{I_i} + L_i$ and $\underset{s'}{R_i} = \underset{s'}{I_i} + L_i$, then we can replace $\underset{s}{R_i}$ and $\underset{s'}{R_i}$ by $\underset{s}{I_i}$ and $\underset{s'}{I_i}$, respectively in the equation. And by generalizing the equation, we add interferences in the set $\underset{s}{\mathbb{S}}_i^D$ (corresponding to τ_k and τ_l) and subtract interferences in the set $\underset{s'}{\mathbb{S}}_i^D \cap \underset{s}{\mathbb{S}}_i^D$ (corresponding to τ_k). We, hence, get:

$$\underset{s}{I_i} = \underset{s'}{I_i} + \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s}{R_i} + J_j^R}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D \cap \underset{s'}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s'}{R_i} + J_j^R}{T_j} \right\rceil * L_j$$

□

Next, we show a theorem for computing the WCL of a task on its path δ_i under the assumption that $D_i \leq (T_i - J_i^R)$ and using only direct interferences. Note that if a task τ_j directly interferes with the task under analysis τ_i on a non-consecutive set of stages, then τ_j must be split into two or more directly interfering tasks such that each newly created task interferes with τ_i on consecutive stages.

Theorem 1. *The WCL R_i of a task τ_i suffering only direct interferences along its path δ_i where the last stage on the path is $s = s_{i,|\delta_i|}$ and under the condition $D_i \leq (T_i - J_i^R)$ is given by:*

$$R_i = \underset{s}{R_i} + J_i^R + \sum_{l=2...|\delta_i|} R_{s_{i,l}}$$

where

$$\underset{s}{R_i} = \underset{s}{I_i} + L_i$$

$$\underset{s}{I_i} = \underset{s'}{I_i} + \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s}{R_i} + J_j^R}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}}_i^D \cap \underset{s'}{\mathbb{S}}_i^D} \left\lceil \frac{\underset{s'}{R_i} + J_j^R}{T_j} \right\rceil * L_j$$

Proof: From Lemma 3, the WCL of a task under analysis τ_i is monotonically increasing with respect to the stages. Hence, the WCL $\underset{s}{R_i}$ is largest at the last stage of the path $s = s_{i,|\delta_i|} \in \delta_i$. The WCL $\underset{s}{R_i}$ is measured from the time t_0 (the release of the τ_i job). Since, the job could have been released after suffering maximum jitter, then the WCL measured from the activation time is equal to $\underset{s}{R_i} + J_i^R$. And since we want to find the WCL across the whole path δ_i, then we must take into account the stage delay $R_{s_{i,l}}$. Since this delay is experienced between stages, then the total delay is equal to $\sum_{l=2...|\delta_i|} R_{s_{i,l}}$. Therefore, the WCL of task τ_i, $R_i = \underset{s}{R_i} + J_i^R + \sum_{l=2...|\delta_i|} R_{s_{i,l}}$ where $s = s_{i,|\delta_i|} \in \delta_i$ is the last stage on the path δ_i. The recurrence relation derived in Lemma 4 is used to find $\underset{s}{R_i}$. □

5.3 Indirect Interference

Task τ_i suffers indirect interference on a stage from task τ_k when task τ_i has direct interference with an intermediate task τ_j, and τ_j has direct interference with task τ_k; however, τ_i has no direct interference with τ_k. In addition, the interference between τ_j and τ_k must occur before τ_j interferes with τ_i. We formally describe this in Definition 6, and the set of indirectly interfering tasks in Definition 7. Revisiting Figure 1a, we point out that task τ_6 has indirect interference with task τ_1 through an intermediate task τ_2 on stage (v_3, v_4) such that $\underset{(v_3,v_4)}{\mathbb{S}}_6^I = \{\tau_1\}$.

Definition 6. *Given two stages s and $\hat{s}$ on path δ_j, task τ_i suffers indirect interference from task τ_k on stage s if and only if $(s \in \delta_i \cap \delta_j) \wedge (\hat{s} \in \delta_j \cap \delta_k) \wedge (s \neq \hat{s}) \wedge (\hat{s} \notin \delta_i) \wedge (\hat{s} \in \sigma_j(s))$, and $P_i < P_j < P_k$.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

Definition 7. *Given two stages s and $\hat{s}$ on path δ_j, the set of tasks indirectly interfering with task τ_i on stage s is $\mathbb{S}_{s}^{I}{}_{i} = \{\tau_k \mid \forall \tau_j, \tau_k \in \Gamma, (s \in \delta_i \cap \delta_j) \wedge (\hat{s} \in \delta_j \cap \delta_k) \wedge (s \neq \hat{s}) \wedge (\hat{s} \notin \delta_i) \wedge (\hat{s} \in \sigma_j(s))$, and $P_i < P_j < P_k\}$.*

Indirect interferences are accounted for by *indirect interference jitter*. Lemma 1 introduced the conditions under which a higher priority task τ_j causes maximum interference to τ_i. It is clear that the interference caused by τ_k can delay the release of the jobs of τ_j. Assume that the interference that τ_j suffers from task τ_k is equal to I_j. From the point of view of τ_i, the maximum jitter that τ_j can suffer is equal to the sum of the interference that τ_j suffers from τ_k and its release jitter; $I_j + J_j^R$. In the worst case (with respect to τ_i), this indirect interference can further delay the jobs of τ_j activated before t_0 while the jobs activated after t_0 are immediately released. This extra jitter caused by the indirect interference is known as *indirect interference jitter*. Consider the timeline in Figure 2. Task τ_6 suffers direct interference on stage $s_{6,2}$ from task τ_2 and suffers indirect interference from task τ_1 through the intermediate task τ_2. Task τ_2 suffers an interference of 2 time units from task τ_1. Hence, the first release of task τ_2 on stage $s_{2,2}$ at $t_0 = 1$ has a maximum indirect interference jitter of 2 time units. The subsequent jobs of task τ_2 are released immediately at their activation time.

It is important to mention that the indirect interference jitter depends on the indirect interference set of the task under analysis. Using the same aforementioned tasks as an example, this means that I_j only contains interference from common tasks in the direct interference set of τ_j and the indirect interference set of τ_i. This is because other tasks in the direct interference set of τ_j, $\mathbb{S}_{s}^{D}{}_{j}$, but not in the indirect interference set of τ_i, $\mathbb{S}_{s}^{I}{}_{i}$, are tasks that directly interfere with τ_i, i.e., in the set $\mathbb{S}_{s}^{D}{}_{i}$. These tasks are already accounted for by direct interference on τ_i and should not be accounted for by indirect interference jitter through τ_j. Hence, the indirect interference that τ_i suffers through τ_j is from tasks in the set $\mathbb{S}_{s}^{D}{}_{j} \cap \mathbb{S}_{s}^{I}{}_{i}$. We use the terms $\underset{s}{R}_j(\tau_i)$ and $\underset{s}{I}_j(\tau_i)$ to denote that WCL and worst-case interference of task τ_j, respectively, only due to tasks in the set $\mathbb{S}_{s}^{D}{}_{j} \cap \mathbb{S}_{s}^{I}{}_{i}$. Therefore, the indirect interference jitter of a task τ_j with respect to a task under analysis τ_i is given by:

$$\underset{s}{J}_j^I = \underset{s}{R}_j(\tau_i) - L_j \tag{2}$$

where

$$\underset{s}{R}_j(\tau_i) = \underset{s}{I}_j(\tau_i) + L_j$$

$$\underset{s}{I}_j(\tau_i) = \underset{s'}{I}_j(\tau_i) + \sum_{\forall \tau_k \in \mathbb{S}_{s}^{D}{}_{j} \cap \mathbb{S}_{s}^{I}{}_{i}} \left\lceil \frac{\underset{s}{R}_j(\tau_i) + J_k^R + \underset{s}{J}_k^I}{T_k} \right\rceil * L_k - \sum_{\forall \tau_k \in \mathbb{S}_{s}^{D}{}_{j} \cap \mathbb{S}_{s'}^{D}{}_{j} \cap \mathbb{S}_{s}^{I}{}_{i}} \left\lceil \frac{\underset{s'}{R}_j(\tau_i) + J_k^R + \underset{s}{J}_k^I}{T_k} \right\rceil * L_k$$

5.4 Worst-case Latency with Indirect Interference

We can now extend Theorem 1 to account for both direct and indirect interferences. The WCL of task τ_i when it experiences both direct and indirect interferences is given by Theorem 2.

Theorem 2. *The WCL R_i of a task τ_i suffering both direct and indirect interferences along its path δ_i where the last stage on the path is $s = s_{i,|\delta_i|}$ and under the condition $D_i \leq (T_i - J_i^R)$ is given by:*

$$R_i = \underset{s}{R}_i + J_i^R + \sum_{l=2\ldots|\delta_i|} R_{s_{i,l}}$$

where

$$\underset{s}{R}_i = \underset{s}{I}_i + L_i$$

$$\underset{s}{I}_i = \underset{s'}{I}_i + \sum_{\forall \tau_j \in \mathbb{S}_{s}^{D}{}_{i}} \left\lceil \frac{\underset{s}{R}_i + J_j^R + \underset{s}{J}_j^I}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \mathbb{S}_{s}^{D}{}_{i} \cap \mathbb{S}_{s'}^{D}{}_{i}} \left\lceil \frac{\underset{s'}{R}_i + J_j^R + \underset{s}{J}_j^I}{T_j} \right\rceil * L_j$$

Proof: Indirect interference jitter delays the release of higher priority jobs directly interfering with the task under analysis τ_i. From Lemma 1, the interval of time during which τ_j can interfere with τ_i is the WCL of the job plus the maximum jitter of τ_j. The maximum jitter of τ_j is equal to the sum of the release and indirect interference jitters of τ_j, $J_j^R + \underset{s}{J}_j^I$. Hence, the interval of time during which τ_j can interfere with τ_i is equal to $\underset{s}{R}_i + J_j^R + \underset{s}{J}_j^I$. Substituting this term for the numerators of the summations in Theorem 1 provides the WCL for task τ_i with direct and indirect interferences. □

5.5 An Illustrative Example

We use Figure 1a as an illustrative example to show how SLA is applied and compare it to FLA [2], [5]. Recall that the tasks in Figure 1a have the data in Table 1d, and the following priorities: $P_1 > P_2 > P_3 > P_4 > P_5 > P_6$. Table 1 shows the WCLs from FLA and SLA for all the tasks. We observe that the results from SLA are less than or equal to the upper- bounds computed by FLA. Task τ_6 is unschedulable using FLA.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

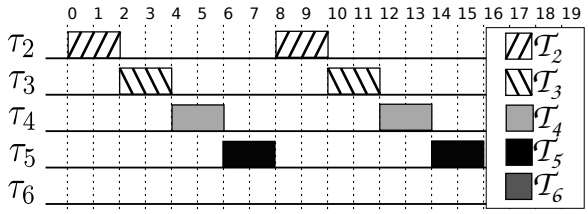


Fig. 4: Timeline of task τ_6 in Figure 1a using FLA.

According to FLA, τ_6 has direct interference with the tasks τ_2, τ_3, τ_4, and τ_5. Figure 4 shows the timeline for task τ_6 using FLA. The four tasks that interfere with τ_6 have a basic stage latency of two time units and period 8. This means that all four tasks consume all bandwidth, i.e. the utilization of the communication resources exceeds a 100%, and the data units of τ_6 can never be sent. Thus, τ_6 is not schedulable for any deadline. This occurs because FLA assumes that τ_6 suffers simultaneous interference from τ_2, τ_3, τ_4 and τ_5 on all stages along its path. This is certainly not the case as shown in Figure 2. In fact, the WCL of τ_6 can be computed, but it requires performing the analysis at the stage-level. This provides a simple example where SLA performs better than FLA.

Task	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
FLA	3	5	3	3	6	-
SLA	3	5	3	3	5	43

TABLE 1: Analysis results.

5.6 Tightness Analysis

We expect SLA to have tighter latency bounds compared to FLA. The reason is that FLA assumes that the interference that a task suffers on any stage occurs on the whole path of the task while SLA restricts the interference only to the stages on which they happen. In what follows, we formally prove that SLA provides tighter bounds than FLA.

Theorem 3. *Given a set of tasks* Γ *and their paths,* $R_i^{SLA} \leq R_i^{FLA},\ \forall \tau_i \in \Gamma$.

Proof: According to FLA [2], the WCL of τ_i measured from the release of τ_i is given by:

$$R_i^{FLA} = \sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i^{FLA} + J_j^R + J_j^I}{T_j} \right\rceil * C_j + C_i \quad (3)$$

The worst-case interference occurs when the higher priority tasks share all stages with the path δ_i of τ_i. This means that the set of all interfering tasks on δ_i, $S_i^D = \mathbb{S}_{\underset{s}{i}}^D$. In that case, using Theorem 2, $\underset{s}{R_i}$ on all stages of δ_i are equal and $R_i^{SLA} = \underset{s}{R_i} + \sum_{l=2\ldots|\delta_i|} R_{s_{i,l}}$ (measured from the release of τ_i). Given that $C_i = L_i + \sum_{l=2\ldots|\delta_i|} R_{s_{i,l}}$, R_i^{SLA} can be given by:

$$\sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i^{SLA} - \sum_{l=2\ldots|\delta_i|} R_{s_{i,l}} + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + C_i \quad (4)$$

For FLA, $J_j^I = R_j - C_j$ and for SLA $\underset{s}{J_j^I}$ is given by Equation 2. For SLA, the worst-case interference jitter occurs when $\mathbb{S}_{\underset{s}{j}}^D = \mathbb{S}_{\underset{s}{i}}^I$. In that case, $\underset{s}{J_j^I} = \underset{s}{R_j} - L_j = R_j - \sum_{l=2\ldots|\delta_j|} R_{s_{i,l}} - L_j = R_j - C_j$. Therefore, $\underset{s}{J_j^I} = J_j^I$. Taking this into account and comparing Equations 3 and 4, the only difference between FLA and SLA are the terms L_j and C_j, and the stage delay in the summation. Since $C_j = L_j + \sum_{l=2\ldots|\delta_j|} R_{s_{i,l}}$, and $|\delta_j| \geq 1$ then $L_j \leq C_j$. And since the stage delay has a negative sign in the numerator of the summation, therefore, $R_i^{SLA} \leq R_i^{FLA}$ and our analysis gives a tighter bound compared to FLA.

For the case when there is no interference, $R_i^{FLA} = C_i$ and $R_i^{SLA} = L_i + \sum_{l=2\ldots|\delta_i|} R_{s_{i,l}} = R_i^{FLA}$. Since in the presence of interference, $R_i^{SLA} \leq R_i^{FLA}$, and in the absence of interference, $R_i^{SLA} = R_i^{FLA}$, then $R_i^{SLA} \leq R_i^{FLA}$. □

5.7 Relaxing Deadline Restriction

In this Section, we relax the assumption $D_i \leq (T_i - J_i^R)$. This means that jobs of the same task can interfere with one another. Since our model transmits data units of the same priority based on FIFO ordering, then a job released earlier in time will have a higher priority than other jobs of the same task that are released later. We use the term *level-i busy period*, $\underset{s}{B_i}$, to define an interval of time during which data units of priority P_i or higher are continuously transmitted on stage s before the stage is idle. For example, in Figure 2, the time interval $t = 4$ to $t = 43$ on stage $s_{6,5} = (v_6, v_7)$ of task τ_6 represents a level-6 busy period. If we can find an upper-bound on a level-i busy period, then we can find the maximum number of jobs of task τ_i that interfere with each other. The WCL of task τ_i is then computed as the maximum WCL of all τ_i jobs in the level-i busy period. First, we find an upper-bound for a level-i busy period. Then, we define the WCL of each τ_i job in the level-i busy period.

Lemma 5. *The level-i busy period on stage* $s \in \delta_i$ *considering both direct and indirect interferences to task* τ_i *is given by:*

$$\underset{s}{B_i} = \underset{s'}{B_i} + \sum_{\forall \tau_j \in \mathbb{S}_{\underset{s}{i}}^D} \left\lceil \frac{\underset{s}{B_i} + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + \left\lceil \frac{\underset{s}{B_i} + J_i^R}{T_i} \right\rceil * L_i - \sum_{\forall \tau_j \in \mathbb{S}_{\underset{s}{i}}^D \cap \mathbb{S}_{\underset{s'}{i}}^D} \left\lceil \frac{\underset{s'}{B_i} + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j - \left\lceil \frac{\underset{s'}{B_i} + J_i^R}{T_i} \right\rceil * L_i$$

Proof: **Base Case:** The rules in Lemma 1 still hold after relaxing the deadline assumption. Considering a level-i busy period, Lemma 1 holds for all tasks of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

priority i or higher. First, we consider the first stage $s = s_{i,1}$ on the path δ_i of the task under analysis τ_i. On this stage, the length of the busy period is the latency of all τ_i jobs plus interference from any higher priority jobs. Assuming the upper-bound on the busy period is $\underset{s}{B_i}$, then according to Lemma 1, the interval of time during which jobs of τ_i can exist in the busy period is equal to the length of the busy period plus the maximum jitter of τ_i, i.e., $\underset{s}{B_i} + J_i^R$. Also, the interval time during which jobs of a higher priority task τ_j exist in the busy period is equal to the busy period length plus maximum jitter, i.e., $\underset{s}{B_i} + J_j^R + \underset{s}{J_j^I}$. Therefore, considering all higher priority tasks, the length of the busy period on the first stage $s = s_{i,1}$ can be given by:

$$\underset{s}{B_i} = \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}_i^D}} \left\lceil \frac{\underset{s}{B_i} + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + \left\lceil \frac{\underset{s}{B_i} + J_i^R}{T_i} \right\rceil * L_i \quad (5)$$

Equation 5 represents the base condition for our proof. We proceed to prove this Lemma by induction, assuming that the given Lemma holds for a stage $s' \in \delta_i$. We showed in Lemma 3 that the latency of task τ_i is monotonically increasing with respect to the stages of the path δ_i. Similarly, the length of the level-i busy period is monotonically increasing with respect to the stages of the path δ_i. We also showed that common higher priority jobs between stages s' and s cannot cause more interference on stage s than the interference they caused on stage s'. Following the same reasoning, jobs of priority i or higher that exist in the busy period of stage s' cannot contribute more to the busy period on stage s than their contribution on stage s'. Hence, we can find the busy period on stage s by adding the latency of all jobs of priority i or higher on stage s to the length of the busy period on s' while subtracting the latency of common jobs.

Since, the length of the busy period is monotonically increasing, then the number of jobs of task τ_i or any higher priority task τ_j on stage s is larger than the number of jobs of the same task on stage s'. The latency of the common jobs between stages s' and s is the latency of the higher priority jobs on stage s' of the tasks in the set $\underset{s}{\mathbb{S}_i^D} \cap \underset{s'}{\mathbb{S}_i^D}$ plus the latency of the jobs of task τ_i that exist on stage s'. This can be represented by:

$$\sum_{\forall \tau_j \in \underset{s}{\mathbb{S}_i^D} \cap \underset{s'}{\mathbb{S}_i^D}} \left\lceil \frac{\underset{s'}{B_i} + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + \left\lceil \frac{\underset{s'}{B_i} + J_i^R}{T_i} \right\rceil * L_i \quad (6)$$

Therefore, we can find the busy period on stage s by summing contribution from jobs of priority i or higher on stage s (as represented by Equation 5) to the length of the busy period on stage s', $\underset{s'}{B_i}$, and subtracting the contribution from common jobs (as represented by Equation 6). Thus, proving the Lemma. □

So far, we have derived an upper-bound on the level-i busy period on a stage s, $\underset{s}{B_i}$. To find the WCL of τ_i on a stage s, we need to compare the WCLs of all jobs in the busy period $\underset{s}{B_i}$. Thus, we need to find the maximum number of jobs of τ_i in the busy period $\underset{s}{B_i}$, $\underset{s}{p_{B,i}} = \left\lceil \frac{\underset{s}{B_i} + J_i^R}{T_i} \right\rceil$. We number the jobs of τ_i in the busy period $\underset{s}{B_i}$ from $p = 1$ to $p = \underset{s}{p_{B,i}}$, with $p = 1$ being the first job released in the busy period. In Theorem 4, we find the WCL of each job in the busy period and use them to find the WCL of τ_i.

Theorem 4. *The WCL R_i of a task τ_i suffering both direct and indirect interferences along its path δ_i where the last stage on the path is $s = s_{i,|\delta_i|}$ is given by:*

$$R_i = \max_{p=1 \ldots \underset{s}{p_{B,i}}} (\underset{s}{w_i}(p) - (p-1) * T_i + J_i^R) + \sum_{l=2 \ldots |\delta_i|} R_{s_{i,l}}$$

where

$$\underset{s}{w_i}(p) = \underset{s}{I_i}(p) + p * L_i$$

$$\underset{s}{I_i}(p) = \underset{s'}{I_i}(p') + \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}_i^D}} \left\lceil \frac{\underset{s}{w_i}(p) + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j - \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}_i^D} \cap \underset{s'}{\mathbb{S}_i^D}} \left\lceil \frac{\underset{s'}{w_i}(p') + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j$$

and

$$p' = \begin{cases} p & \text{if } p \leq \underset{s'}{p_{B,i}} \\ \underset{s'}{p_{B,i}} & \text{otherwise} \end{cases} \quad (7)$$

Proof: We first derive $\underset{s}{w_i}(p)$, the worst-case completion time of job p on stage s measured from the start of the busy period $\underset{s}{B_i}$, on the first stage $s = s_{i,1}$. Then proceed to prove the Theorem by induction.

Base Case: The time interval $\underset{s}{w_i}(p)$ represents the time at which the p^{th} job of τ_i completes transmission. It includes higher priority jobs of interfering tasks and higher priority jobs of τ_i (jobs that were released earlier than job p). The latency of τ_i jobs in $\underset{s}{w_i}(p)$ including the p^{th} job is equal to $p * L_i$. The interval during which a higher priority task τ_j can interfere with τ_i is equal to the time interval $\underset{s}{w_i}(p)$ plus maximum jitter, i.e., $\underset{s}{w_i}(p) + J_j^R + \underset{s}{J_j^I}$. Hence, considering all higher priority tasks, we can represent the time interval $\underset{s}{w_i}(p)$ on the first stage $s = s_{i,1}$ by:

$$\underset{s}{w_i}(p) = \sum_{\forall \tau_j \in \underset{s}{\mathbb{S}_i^D}} \left\lceil \frac{\underset{s}{w_i}(p) + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + p * L_i \quad (8)$$

Notice that the definition of $\underset{s}{w_i}(p)$ reduces to Equation 8 on the first stage of the path δ_i. Equation 8 serves as the base condition for our proof. We now assume the Theorem holds for a stage $s' \in \delta_i$ and derive it for stage s.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers

Again, we use two facts that we proved in Lemma 3. The first is that $\underset{s}{w_i}(p)$ is monotonically increasing. And the second is that common higher priority jobs between stages s' and s cannot cause more interference on stage s. Assuming that job p exists on both stages s and s', then we can obtain $\underset{s}{w_i}(p)$ by adding to $\underset{s'}{w_i}(p)$ the latency of jobs that exist on stage s and subtracting the latency of common jobs between stages s and s'. Note that the common jobs include jobs from τ_i as well. Hence, we can represent $\underset{s}{w_i}(p)$ by:

$$\underset{s}{w_i}(p) = \underset{s'}{w_i}(p) + \sum_{\forall \tau_j \in \mathbb{S}_i^D} \left\lceil \frac{\underset{s}{w_i}(p) + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + p * L_i - \sum_{\forall \tau_j \in \mathbb{S}_i^D \cap \mathbb{S}_{s'i}^D} \left\lceil \frac{\underset{s'}{w_i}(p) + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j - p * L_i \quad (9)$$

Equation 9 is valid under the assumption that job p of task τ_i exists on both stages s' and s. Since $\underset{s}{w_i}(p)$ is monotonically increasing, then the number of τ_i jobs on stage s are greater than or equal to that on stage s'. Thus, job p might only exist on stage s but not s'. So we introduce the term p'. If job p exists on s', then $p' = p$ and Equation 9 holds. However, if job p does not exist on stage s', then $\underset{s'}{w_i}(p)$ is undefined. In such case, we find common jobs in the interval $\underset{s'}{w_i}(\underset{s'}{p_{B,i}})$ which is the worst-case completion time of the last job $\underset{s'}{p_{B,i}}$ on stage s' measured from the start of the busy period $\underset{s'}{B_i}$. This is effectively the whole length of the busy period on stage s'. Therefore, we modify Equation 9 to use p' to refer to the job on the preceding stage s'. The equation thus becomes:

$$\underset{s}{w_i}(p) = \underset{s'}{w_i}(p') + \sum_{\forall \tau_j \in \mathbb{S}_i^D} \left\lceil \frac{\underset{s}{w_i}(p) + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j + p * L_i - \sum_{\forall \tau_j \in \mathbb{S}_i^D \cap \mathbb{S}_{s'i}^D} \left\lceil \frac{\underset{s'}{w_i}(p') + J_j^R + \underset{s}{J_j^I}}{T_j} \right\rceil * L_j - p' * L_i \quad (10)$$

Since $\underset{s}{w_i}(p) = \underset{s}{I_i}(p) + p * L_i$ and $\underset{s'}{w_i}(p') = \underset{s'}{I_i}(p') + p' * L_i$, then substituting them in Equation 10 yields the definition of $\underset{s}{w_i}(p)$ in the Theorem.

Now, we need to find the WCL of each of the jobs of τ_i in the busy period $\underset{s}{B_i}$. Since $\underset{s}{w_i}(p)$ is the worst-case completion time of job p in the interval $\underset{s}{B_i}$, where $s = s_{i,|\delta_i|}$ is the last stage on path δ_i then subtracting the activation time of job p from $\underset{s}{w_i}(p)$ yields the WCL of the job. The first job $p = 1$ is released at the start of the busy period after suffering maximum release jitter, hence the activation time of the first job is at time $-J_i^R$ (relative to the start of the busy period). The second job is released immediately at its activation time, i.e., at time $T - J_i^R$, and the third job at time $2 * T - J_i^R$. Hence, the p^{th} job is activated at time $(p-1) * T_i - J_i^R$. The WCL of job p is thus $\underset{s}{w_i}(p) - (p-1) * T_i + J_i^R$. Taking the maximum latency across all jobs, $p = 1 \ldots \underset{s}{p_{B,i}}$, yields the WCL of task τ_i on the last stage of its path δ_i. Adding the stage delay to WCL gives us the WCL of τ_i along its path δ_i:

$$R_i = \max_{p=1 \ldots \underset{s}{p_{B,i}}} (\underset{s}{w_i}(p) - (p-1) * T_i + J_i^R) + \sum_{l=2 \ldots |\delta_i|} R_{s_{i,l}}$$

□

6 EXPERIMENTATION

We quantitatively evaluate the proposed stage-level analysis. The evaluation is performed on a priority-aware NoC with flit-level preemption [2]. A communication packet consists of smaller data units called flits. Each node in the NoC consists of a processing element and a router. Priority-aware routers have multiple virtual channels (VCs) with associated priorities. These priorities are used to preempt the routing of packets at the flit level in lower priority VCs by flits in higher priority VCs. The router selects the output port for a data unit in the VCs based on its desired destination. Computation tasks execute on the processing elements (processing resources) and communicate using messages on the NoC links. Messages transmitted over the network links map to communication tasks executed on communication resources in our model. Data is transmitted in parallel across the links between two communicating processing elements through wormhole switching. Both SLA and FLA can be applied to the priority-aware NoC.

The quantitative evaluation for SLA is performed on a large set of synthetic benchmarks. Using synthetic benchmarks allows varying different factors and measuring their effect on SLA. It also allows us to test extremes of factors, e.g. high and low utilizations, that otherwise are difficult to test for. We perform our experiments on 4×4 and 8×8 instances of the NoC. We compare the analytical bounds of SLA to those of FLA [2]. We randomly generate communication tasks and compute the WCLs using both SLA and FLA.

The goals of our experimentation are as follows:

- Show that SLA schedules more task sets compared to FLA, and quantify the increase in schedulability.
- Quantify the improvement (tightness) of the latency bounds computed by SLA over FLA.
- Quantify the percentage of tasks of schedulable task sets that result in an improved latency bound.
- Compare the analysis time of SLA and FLA.

The experiment setup involves changing a number of factors to assess their effect on the analysis.

1) Number of communications tasks is in the range (1,100) in steps of 1.
2) Task period (or minimum interarrival time for sporadic tasks), T_i, is chosen in the range (1000,1000000) through a uniform random distribution.

3) Task deadline, D_i, is chosen as a multiple of the period, e.g., $2 * T_i$.
4) Communication utilization is varied in the range (10%, 6000%) in steps of 60%.This factor represents the utilization of the communication resources in the network. Full utilization of a single communication resource is represented by 100% utilization. The full utilization of a 4×4 mesh is represented by 2400%
5) Task release jitter, J_i^R, is set to zero.
6) An arbitrary priority assignment scheme is used for choosing task priorities.
7) Task mapping is random.
8) A shortest path algorithm is used to select paths for communication tasks between source and destination nodes.
9) A 100 test cases are generated for each possible configuration (40000 configurations).

We use the following metrics for evaluation:

- **Schedulability:** A communication task is unschedulable if: 1) its total WCL is larger than its deadline ($R_i > D_i$), or 2) no solution is found for the iterative analysis equation on any of the stages on its path. For the latter case, the stopping condition for the equation on any stage is when the latency computed in an iteration exceeds the deadline. A test case will be unschedulable if one of the tasks in its task set is unschedulable. The schedulability metric is a measure of the percentage of schedulable test cases for a particular configuration.
- **Improvement in WCLs:** For each test case, we compute the WCL of each task using both SLA and FLA. For a particular communication task, the improvement in the computed WCL bound is calculated as $1 - R_i^{SLA}/R_i^{FLA}$. We report the average improvement for a test configuration. This metric is only valid for schedulable tasks.
- **Fraction of tasks with improved latencies:** For a given test case, this metric shows the percentage of tasks that have a tighter WCL bound using SLA compared to FLA. This metric only applies to schedulable tasks.
- **Analysis time:** This is the time taken to compute the latency bounds for all tasks in a test case using both SLA and FLA. For any given configuration, we report the average analysis time over all test cases.

Schedulability: Figures 5a and 5b show the schedulability against the communication utilization for task sets with 100 tasks and deadlines of $2 * T$ and $10 * T$, respectively. Both graphs demonstrate approximately the same trend. For a 4×4 NoC instance, both SLA and FLA are able to schedule task sets for very low utilizations. As the utilization increases, SLA schedules more task sets compared to FLA. The reason is that SLA performs the analysis on the stage-level, thus reducing the latency bounds and increasing the schedulability of task sets. The same behavior is observed for an 8×8 NoC instance. However, the gap in schedulability between SLA and FLA widens compared to the 4×4 instances. This is because 8×8 NoC instances have more communication resources and tasks can traverse longer paths which leads to more interferences. Also higher communication utilizations means that tasks have larger basic stage latencies which leads to larger interferences. As interferences increase, the tightness of SLA manifests more and leads to a larger schedulability gap compared to FLA.

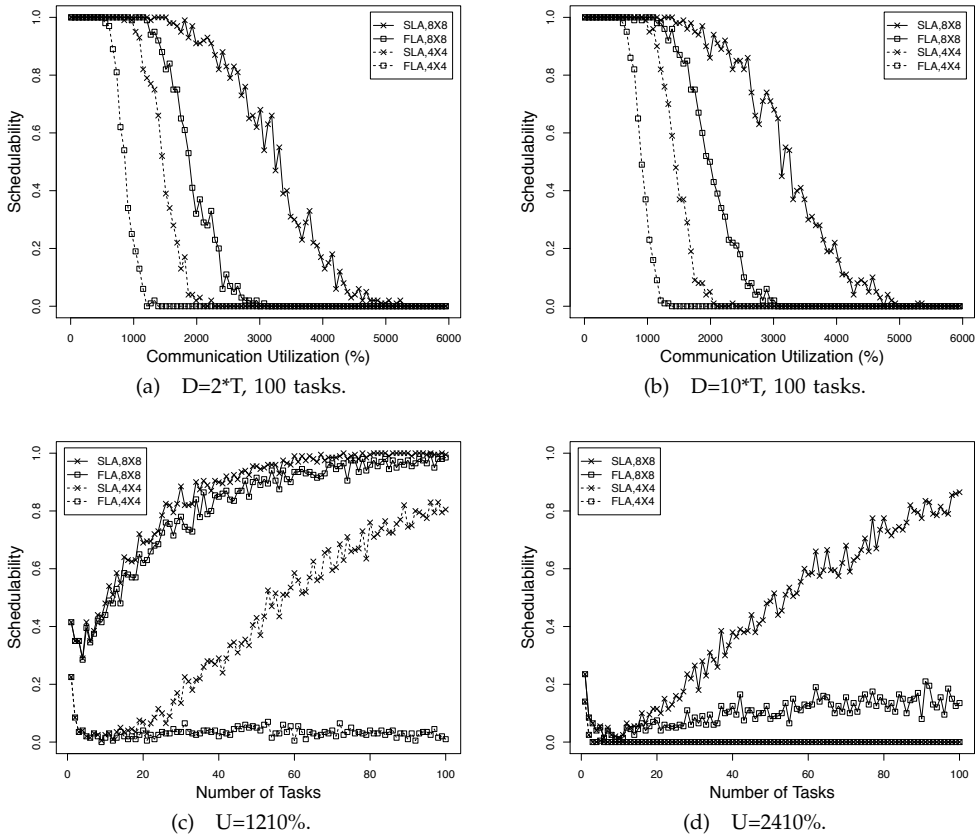
Figures 5c and 5d show the schedulability against the number of tasks for utilizations 1210% and 2410%, respectively. At a communication utilization of 1210%, SLA performs slightly better than FLA in terms of schedulability in 8×8 NoC instances. As the number of tasks increase, the ratio of schedulable task sets increases for both SLA and FLA. The reason is that the utilization is divided amongst more tasks, thus reducing the per task utilization and accordingly interferences, hence, scheduling more task sets. For a 4×4 NoC instance, SLA schedules more task sets than FLA which has a very low schedulability ratio even as the number of tasks increases. The reason is that at $U = 1210\%$ in 4×4 NoC instances, FLA is still not able to schedule all tasks due to the high interferences while SLA can perform better because it provides a tighter analysis. For a communication utilization of 2410%, both SLA and FLA cannot schedule any of the task sets with more than 10 tasks in 4×4 NoC instances due to high interferences. In an 8×8 NoC instance, SLA schedules more task sets compared to FLA as the number of tasks increases. This further demonstrates the tightness of SLA over FLA.

Response Time: Figure 6a shows the latency improvement against the communication utilization for task sets with 100 tasks. As the utilization increases, the improvement in latencies computed by SLA over FLA increases to about 15%. The reason is that increasing the utilization, increases the interferences between tasks, leading to a wider gap in the computed latency bounds between SLA and FLA. This happens up to a certain utilization turning point, after which the improvement starts decreasing again as the utilization increases. This is because beyond that turning point, more interferences cause more tasks to be unschedulable. This turning point is at a higher utilization value in 8×8 NoC instances due to the existence of more communication resources.

Figure 6b shows the latency improvement against the number of tasks for a communication utilization of 3610%. Increasing the number of tasks, increases the interferences, thus, leading to a higher improvement in latency bounds. The improvement in latency is higher in 8×8 NoC instances because more tasks can be scheduled due to the presence of more communication resources.

Figure 6c shows the ratio of tasks with improved latency bound against the number of tasks at a communication utilization of 3610%. As the number of tasks increases, more interference occurs, and more tasks have an improved latency bound using SLA compared to FLA. The percentage of tasks with improved latency

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers



(a) D=2*T, 100 tasks. (b) D=10*T, 100 tasks.

(c) U=1210%. (d) U=2410%.

Fig. 5: Schedulability results.

increases as the number of tasks increases, and reaches approximately a 100%.

Computation Time: Figure 6d compares the average computation times of both SLA and FLA. The analysis time for SLA is approximately double that of FLA. We find this to be reasonable for the quality of results delivered by SLA. This is acceptable since SLA performs the analysis at the stage-level compared to the flow-level using FLA.

Summary: For all $4,000,000$ test cases, the WCL is reduced on average by 5.2%, and the number of schedulable test cases is increased by 34.0%. Any task that is schedulable using FLA is also schedulable using SLA. The ratio of latency bounds (SLA to FLA) is less that or equal 1.0 for all schedulable tasks. This means that SLA is at worst the same as FLA, which verifies our tightness analysis. The analysis time of SLA is on average double that of FLA. The average analysis time over all test cases for SLA is 71.0 mS and 36.4 mS for FLA. We use the Wilcoxon matched pairs test to reason about the significance of our results. The p-value for both the analysis, and the schedulability is less than 2.2×10^{-16}.

7 Conclusion

This work presents a stage-level WCL analysis for pipelined communication interconnects. The proposed SLA accounts for direct and indirect interference that arise from interferences between tasks. We analytically show that SLA will in the worst-case provide results that are equivalent to that of FLA, but otherwise provide tighter estimates. We illustrate this with an example that for a fixed topology, task mapping, and their respective paths, the number of schedulable tasks when using SLA is higher than FLA. Our results show an average improvement over FLA in the WCL analysis by approximately 5% and in schedulability of tasks by 34%. Compared to FLA, SLA can also be used for path selection for flows in the network to further improve schedulability.

References

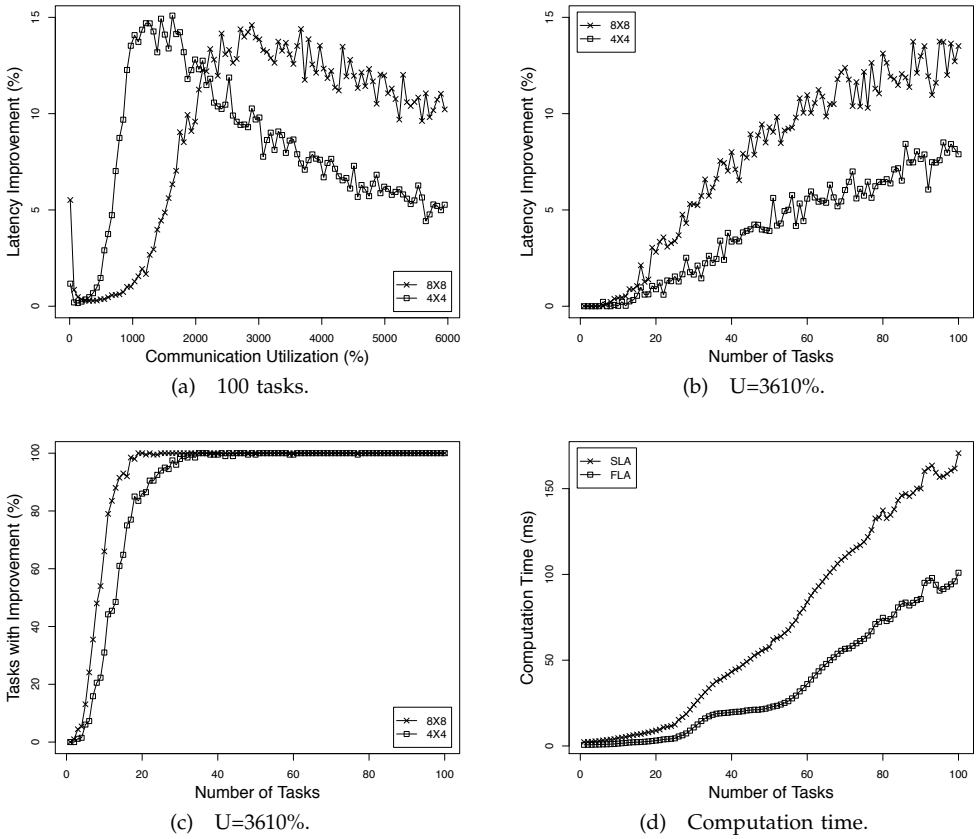
[1] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design and Test*, vol. 22(5), pp. 414 – 421, 2005.

[2] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '08, 2008, pp. 161–170.

[3] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in *Proceedings of the conference on Design, automation and test in Europe - Volume 2*, ser. DATE'04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 20 890–.

[4] Z. Lu and A. Jantsch, "TDM Virtual-Circuit Configuration for Network-on-Chip," *IEEE Transactions on Very Large Scale Integrated Systems*, vol. 16, pp. 1021–1034, August 2008.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2315617, IEEE Transactions on Computers



(a) 100 tasks. (b) U=3610%.

(c) U=3610%. (d) Computation time.

Fig. 6: Latency and computation time results.

[5] Z. Shi and A. Burns, "Schedulability Analysis and Task Mapping for Real-Time on-Chip Communication," *Real-Time Syst.*, vol. 46, pp. 360–385, December 2010.

[6] H. Kashif, H. D. Patel, and S. Fischmeister, "Using link-level latency analysis for path selection for real-time communication on nocs," in *Proceedings of the Asia South Pacific Design Automation Conference*, Sydney, Australia, February 2012, pp. 499–504.

[7] H. Kashif, S. Gholamian, R. Pellizzoni, H. D. Patel, and S. Fischmeister, "ORTAP:An Offset-based Response Time Analysis for a Pipelined Communication Resource Model," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2013.

[8] H. Kashif and H. Patel, "Bounding buffer space requirements for real-time priority-aware networks," in *Proceedings of the Asia South Pacific Design Automation Conference*, Sydney, Australia, January 2014, pp. 113–118.

[9] J. yves Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, 2001.

[10] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *IEEE International Symposium on Circuits and Systems*, 2000.

[11] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, pp. 117–134, 1994.

[12] J. Maki-Turja and M. Nolin, "Fast and tight response-times for tasks with offsets," in *Euromicro Conference on Real-Time Systems*, 2005, pp. 127–136.

[13] J. Palencia and M. Gonzalez, "Schedulability analysis for tasks with static and dynamic offsets," in *IEEE Real-Time Systems Symposium*, 1998, pp. 26–37.

[14] P. Jayachandran and T. Abdelzaher, "A delay composition theorem for real-time pipelines," in *Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on*, 2007, pp. 29–38.

[15] ——, "End-to-end delay analysis of distributed systems with cycles in the task graph," in *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, 2009, pp. 13–22.

[16] L. Phan, S. Chakraborty, and P. S. Thiagarajan, "A multi-mode real-time calculus," in *Real-Time Systems Symposium, 2008*, 2008, pp. 59–69.

[17] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, "Numerical analysis of worst-case end-to-end delay bounds in fifo tandem networks," *Real-Time Systems*, vol. 48, no. 5, pp. 527–569, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11241-012-9153-1

[18] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*, 1990, pp. 201–209.

[19] J. C. Palencia and M. G. Harbour, "Response time analysis of edf distributed real-time systems," *J. Embedded Comput.*, vol. 1, no. 2, pp. 225–237, Apr. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1233760.1233766

Hany Kashif is a PhD student in the Electrical and Computer Engineering department at the University of Waterloo, Canada. His research interests include real-time embedded systems and runtime instrumentation.

Sina Gholamian received his undergraduate degree from University of Tehran in 2010, and his Master's from University of Waterloo in 2012. He is currently an embedded software architect at Thalmic Labs.

Hiren Patel is an assistant professor in the Electrical and Computer Engineering department at the University of Waterloo, Canada. His research interests are in embedded software and hardware systems. This includes models of computation, real-time embedded systems, computer architecture, and system-level design.