PREPRINT: "H. Kashif, S. Gholamian, R. Pellizzoni, H. Patel, and S. Fischmeister, "ORTAP: An Offset-based Response Time Analysis for a Pipelined Communication Resource Model," in proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013, pp. 247-258."

# ORTAP:An Offset-based Response Time Analysis for a Pipelined Communication Resource Model

Hany Kashif, Sina Gholamian, Rodolfo Pellizzoni, Hiren D. Patel and Sebastian Fischmeister
Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada
{hkashif, sgholamian, rpellizz, hdpatel, sfischme}@uwaterloo.ca

***Abstract*—This work addresses the challenge of computing worst-case response times of hard real-time applications deployed on multiprocessor systems. In particular, the worst-case response time analysis (WCRTA) focuses on the communication between distributed tasks of hard real-time applications. The proposed WCRTA models the communication as a pipelined communication resource model. This model incorporates the effect of pipelining, and the parallel transmission of data. Applications of such a model include multiprocessor systems that use complex interconnects such as network-on-chips (NoC)s with priorities. In this paper, we present an exponential analysis, and a polynomial analysis, and prove its correctness. As an application, we apply the pipelined communication resource model to priority-aware NoCs, and we compare the proposed analyses against prior analysis techniques. Our experimental evaluation on two instances of $4 \times 4$ and $8 \times 8$ NoCs with 512,000 synthetic benchmarks shows 48.3% and 66.7% improvement in schedulability for the two NoC sizes over prior work.**

## I. Introduction

Hard real-time applications must guarantee that their temporal requirements are met at all times. This requires a worst-case response time analysis (WCRTA), which provides a method to compute the upper-bounds on the amount of time it takes tasks of an application to complete execution. Such an analysis is essential in determining whether a hard real-time application meets its application deadlines. If it does, then the application is deemed schedulable; otherwise, unschedulable. The requirement to deliver tight worst-case response time (WCRT) estimates is paramount when developing WCRTA techniques because it improves schedulability.

An important challenge in distributing hard real-time systems onto modern computing platforms is in developing WCRTA techniques that combine communication and computation execution latencies. Such WCRTA techniques must consider the worst-case latency of data transmissions across the communication medium connecting the processing resources, and its effect on any dependent computation tasks to determine accurate WCRT estimates. To address this challenge, researchers proposed various WCRTA techniques aiming to provide tight and accurate WCRTs of such distributed hard real-time systems [1], [2], [3]. These efforts make the fundamental assumption that the communication occurs over a single shared bus interconnect. A shared bus interconnect consists of a single communication resource that only allows mutually exclusive access. This presents a traditional communication resource model, but, it does not apply to computing platforms prevalent today. Nowadays, platforms consist of multi-processor systems with multiple processing resources that are typically connected using communication resources such as a network-on-chip (NoC). Modelling the interconnect as a single shared bus interconnect does not accurately model the communication resources available in such platforms. Furthermore, it does not capture the pipelined nature of the communication resources that allow for parallel transmission of data between processing resources across multiple stages of the communication resources. This prohibits accurately predicting the latencies offered by communication resources such as NoCs resulting in gross over-estimates for the WCRTs.

We find that an important challenge in distributing hard real-time applications onto modern computing platforms is in devising WCRTA techniques that incorporate a communication resource model representative of modern communication interconnects. To this end, we propose a WCRTA technique that uses a pipelined communication resource model. We present the theory behind the WCRTA, which includes two variants of the analysis. The first is an exponential analysis, and the second is a polynomial. In general, the exponential analysis is intractable; however, the polynomial analysis is tractable. We present an application of the pipelined communication resource model by constructing two instances of priority-aware NoCs as presented by Shi and Burns [4], [5], [6]. Our experimental evaluation of this application uses a large suite of synthetic benchmarks varying the utilization on the NoC. We contend that using synthetic benchmarks for the NoC application is the correct method for evaluating the strengths and weaknesses of the proposed WCRTA technique. This is because concrete deployments of software provide a restricted exploration space. With synthetic benchmarks, we are able to stress the analysis techniques by varying parameters such as communication interconnect utilization. We compare the proposed WCRTA technique against known prior works including that of Palencia and Gonzalez [2], and a transactional extension of the work by Shi and Burns [4]. Every approximated analysis technique uses the methodology proposed by Maki and Turja [3]. We show application schedulability, and the execution times of the analysis for both the exponential and polynomial WCRTA techniques. The results indicate that the schedulability of ORTAP is 48.3% and 66.7% higher than the schedulability of OFLA and Palencia, respectively.

## II. Related Work

Tindell and Clark [1] introduce a WCRTA for a transactional task model where multiple tasks within the transaction can be

distributed onto multiple processing resources interconnected via a shared bus. Each transaction is a sequence of dependent tasks. The task offsets between task activations guarantee the linear precedence order of task execution. By integrating the notion of offsets, the WCRTA by Tindell and Clark eliminates the pessimism introduced by assuming all tasks are released simultaneously. Hence, they [1] define a new set of conditions for creating the critical instant by considering inter-task offsets. Palencia and Gonzalez [2] extend Tindell and Clark's WCRTA by including dynamic offsets, and allowing them to be larger than the period. Since the exact analysis has exponential time complexity in number of tasks, Palencia and Gonzalez [2] present an approximated version of it that is computable in polynomial time. Later on, Maki and Turja [3] propose an optimization in calculation of WCRTs. The improvement comes from subtracting a new term from the interference function, and that results to less pessimism and tighter response times.

An alternative analysis technique known as delay calculus [7] presents a method to calculate the end-to-end latency of an application. It also uses a pipelined resource model; however, the assumptions and the target for the model are different than ours. In particular, delay calculus requires that the task executes completely before proceeding to the next stage of the pipeline. In our model, we do not have this restriction. We find that by not having this restriction, we can model communication resources such as NoCs that support switching techniques that operate at the flit-level such as wormhole switching.

Our previous work [8] presents an offset-based flow-level analysis (OFLA) for calculating the WCRTs of applications deployed on a multi-processor system based on the analysis by Shi and Burns [4], [5], [6]. Note that our work focuses on the priority-aware NoC proposed by Shi and Burns [4], [5], [6], [9] that uses run-time arbitration as opposed to time-division multiplexing NoCs such as those proposed by Goossens et al. [10]. This work also extends the application model from sequentially dependent tasks to more generalized directed acyclic graph dependencies (DAG). OFLA reaches higher schedulability and tighter response times compared to previous methods. The work presented in [8] is significantly different than the proposed WCRTA in this paper. The primary difference is that OFLA views the pipelined communication resource as a single indivisible resource unit. This means two datums being transmitted on different stages of the same pipeline are said to be interfering with each other. Clearly, this is not an accurate representation of modern interconnects. Our WCRTA in this paper addresses this issue by analyzing interferences at each stage. We compare the proposed analysis in this paper with OFLA as well.

## III. System Model

In this section, we describe the processing and communication resource model, and our task model. We also present an overview of offsets and jitters of tasks.

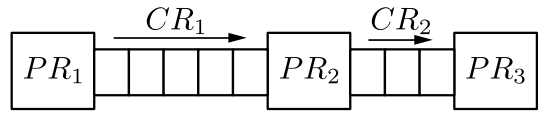



Fig. 1: Illustrative example of processing and communication resource model.

### A. Processing and Communication Resource Model

We use an abstract model to represent the deployment platform consisting of a set of processing resources (PR)s interconnected with pipelined communication resources (CR)s. Naturally, computation tasks execute on computation resources, and communication tasks on communication resources. We assume that such a mapping of tasks to resources is available. Figure 1 illustrates this model. Notice that between two PRs, there are pipelined CRs: one with 5 stages, and another with 3 stages. Data transmitted on the CRs travels through the first stage followed by the next, and so on. Simultaneous transmission of data on stages is allowed. This means that while data is being transmitted on a later stage, new data can be transmitted on an earlier stage in parallel. The communication tasks responsible for transmitting over the CRs (as well as computation tasks executing on the PRs) support fixed priority preemptive scheduling. The transmission latency on each stage is divided into time slots, and we transmit a datum in a time slot. Hence, when a datum transmits in a time slot $t$ on stage $s_1$, it becomes ready for transmission at time $t+1$ on stage $s_2$, and so on. When a datum becomes ready for transmission in a slot $t$, then it will actually be transmitted in that slot unless it is preempted by a higher priority datum. A time slot $t$ is a time interval $[t, t+1)$ (we use both notations interchangeably).

In our model, buffers exist at the processing resources and between stages. We generally assume that there is enough buffer space for storing data. We, however, as a direct result of our WCRTA, can obtain an upper-bound on the required buffer space.

One possible architectural implementation of our proposed interconnect is a worm-hole switched priority-aware network-on-chip with flit-level preemption [4]. Processing resources, in our model map to the network processing elements including the routers, and communication resources map to links. Data transmitted by PRs are broken down into smaller units called flits and each flit is transmitted in one cycle. Parallel transmission of data on multiple links is achieved through worm-hole switching.

### B. Task Model

We model a real-time system as a set of $n$ applications $\mathcal{A} := \{A_1, A_2, ..., A_n\}$ where each application $A_i \in \mathcal{A}$ is denoted by a 4-tuple $\langle G_{A_i}, D_i, T_i, J_i^R \rangle$. The application is a directed-acyclic graph (DAG) with a task graph $G_{A_i} = \langle \Gamma_i^C, \Gamma_i^M \rangle$ consisting of a set of nodes $\Gamma_i^C$ that represent computation tasks and a set of edges $\Gamma_i^M$ that represent communication tasks, respectively. $A_i$ has an end-to-end deadline $D_i$, period $T_i$, and

release jitter $J_i^R$. A computation task of $A_i$, $\tau_{ik} \in \Gamma_i^C$ has a worst-case execution time (WCET) of $C_{ik}$ when executed on some processing resource (PR) $v_{c_{ik}}$, and priority $\mathbb{P}_{ik}$. A communication task of $A_i$, $\tau_{ik} \in \Gamma_i^M$ transmits data across a series of contiguous communication resources (stages) $\delta_{ik}$ from PR $v_{s_{ik}}$ to PR $v_{d_{ik}}$ with priority $\mathbb{P}_{ik}$, and a worst-case transmission latency $L_{ik}$ per communication resource is used. The worst-case transmission latency $L_{ik}$ is the latency that an instance of the task $\tau_{ik}$ takes to transmit data on a single communication resource when it does not suffer interferences from any other tasks. We use the notation $\tau_{ikc}$ to refer to the $c$-th instance (job) of task $\tau_{ik}$. For clarity of presentation, we use the function schedule $\Theta(t, s)$ to denote the assignment of datums of the various jobs to the CR. More specifically, each time slot $t$ on a stage $s$ of the CR is either transmitting a datum or is idle.

Regarding a NoC implementation, computation tasks will execute on the processing elements of the NoC. Communication tasks are messages communicated between the computation tasks. Messages are transmitted on a set of contiguous links that form paths between the processing nodes.

We restrict the task graph to be single rooted, and for it to be a computation task that is activated at the application's period, and has a maximum release jitter $J_i^R$. The release jitter $J_i^R$ is the worst-case delay in the release time of the application or the first task in the application's task graph. The exit task is also a computation task of the task graph without any successor computation tasks. $A_i$ is schedulable if and only if the WCRT of each exit task is less than or equal to $D_i$. Notice that a communication task enforces precedence constraints between other computation and communication tasks. For example, a communication task $\tau_{ik}$ executes only after its source computation task completes execution, and the destination computation task only begins after both the source computation task, and the corresponding communication task completes execution. We do not place any restrictions on the deadlines, the release jitters, and the periods such that the deadline and/or the release jitter can be larger than the period. We assume distinct priority assignment to the tasks, but, we do not enforce any specific priority assignment to the computation and communication tasks, i.e., the priority assignment does not have to follow the order or precedence of the tasks in the task graph $G_{A_i}$. We also refrain from discussing priority sharing for brevity; however, this can be considered an extension to the presented work. Furthermore, the problem of optimally assigning priorities to tasks is orthogonal to this work. Additional details of the proposed task model are available here [8].

### C. Offsets and Jitters

Our task graph represents precedence dependencies between computation tasks through communication tasks. We use offsets and jitters to ensure that these precedence constraints are satisfied. We also support dynamic offsets as introduced by Palencia and Gonzalez [2]. The root task is activated periodically with a period $T_i$. Each task $\tau_{ik}$ in the application is activated after a specific time interval from the activation of the root vertex. We call this time interval, offset $\Phi_{ik}$, which is the best-case release time of task $\tau_{ik}$. This occurs when the preceding tasks execute for their WCET without suffering interferences from higher priority tasks. The offset for the root task is zero. A task's offset is equal to the sum of the WCETs of the tasks along the path leading to that task starting from the root vertex. There might exist multiple paths leading to a task, the offset, in that case, is equal to the maximum offset of all paths leading to that task. If interference exists, then the task release can be delayed from its best-case release time. The release jitter of a task is the maximum difference between its activation time and its release time, i.e., it is the difference between the best-case and the worst-case release times of the task. Again, if multiple paths lead to the task, then the release jitter is equal to the maximum jitter from all paths. The worst-case release time of a task, from the activation of the root vertex, is the sum of its offset $\Phi_{ik}$ and release jitter $J_{ik}^R$. Note that the release jitters of the tasks depend on their WCRTs, and the computation of WCRTs depends on the release jitters. This, therefore, requires iteratively computing WCRTs and assigning jitters until either a fixed point is reached or the application is unschedulable. Figure 2 shows a schedule to illustrate offsets,

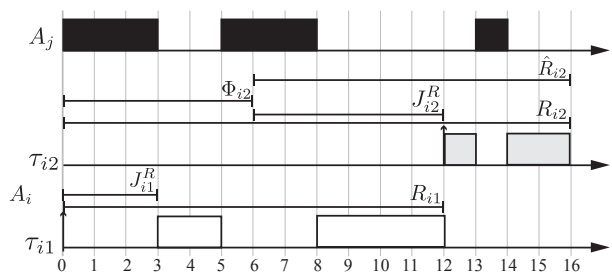



Fig. 2: A schedule to illustrate offsets, jitters, and response times.

jitters, and response times. This example has two applications $A_i$ and $A_j$. $A_i$ has two tasks $\tau_{i1}$ and $\tau_{i2}$ such that $\tau_{i2}$ can only execute after $\tau_{i1}$ completes (dependence). $A_j$ has multiple tasks that all have higher priorities than tasks of $A_i$ such that they cause interference. Up arrows denote release times. Task $\tau_{i1}$ releases at time 0, but it experiences interference from a task in $A_j$ delaying its execution to time 3. Notice that $\tau_{i1}$ releases at time 0; however, it could be released at any time up to and including its release jitter, which is the application's release jitter $J_{i1}^R = J_i^R$. $\tau_{i1}$ suffers another interference from another task of $A_j$ at time 5 causing $\tau_{i1}$ to delay its end time to 12. The worst-case response time $R_{i1}$ of $\tau_{i1}$ is 12. The best-case release time for $\tau_{i2}$ is the WCET of task $\tau_{i1}$ because it is dependent on $\tau_{i1}$ completing its execution. Hence, $\Phi_{i2}$ is 6. $\tau_{i2}$ releases at time 12, which is the end time for $\tau_{i1}$. The difference between the release time of $\tau_{i2}$ and its $\Phi_{i2}$ provides the release jitter $J_{i2}^R$. The WCRT of $\tau_{i2}$ is denoted by $R_{i2}$, which is 16. Note that we denote $\hat{R}_{i2}$ as the WCRT from the task's activation. For $\tau_{i2}$, this is 10.

## IV. ORTAP

ORTAP computes the WCRT for communication tasks by considering the pipelining and parallel data transmission of

jobs on the pipelined CR. We present an exponential and a polynomial analysis. We also prove that each of these analyses gives a safe upper-bound for the WCRT of the communication task under analysis.

### A. Direct and Indirect Interference Sets

Given a task under analysis $\tau_{ab}$, a directly interfering task is a higher priority task $\tau_{ij}$ that preempts the transmission of $\tau_{ab}$ along its CR $\delta_{ab}$. The set of all higher priority tasks that preempt $\tau_{ab}$ along $\delta_{ab}$ is its direct interference set $S^D(\tau_{ab})$. We use the symbols $S_i^D(\tau_{ab})$ and $\underset{s}{S_i^D}(\tau_{ab})$ to denote the directly interfering set of tasks from application $A_i$ along CR $\delta_{ab}$, and on a particular stage $s$, respectively.

*Definition 1:* **Direct interference.** A communication task $\tau_{ab}$ suffers direct interference from task $\tau_{ij}$ on stage $s$ of its CR $\delta_{ab}$ if and only if $s \in \delta_{ab} \cap \delta_{ij} \wedge (P_{ab} < P_{ij})$.

*Definition 2:* **Direct interference set.** The set of communication tasks from application $A_i$ directly interfering with $\tau_{ab}$ on stage $s$ of its CR $\delta_{ab}$ is $\underset{s}{S_i^D}(\tau_{ab}) = \{\tau_{ij} \mid \forall \tau_{ij} \in \Gamma_i^M, s \in \delta_{ab} \cap \delta_{ij} \wedge (P_{ab} < P_{ij})\}$.

Task $\tau_{ab}$ suffers indirect interference from task $\tau_{kl}$ when task $\tau_{ab}$ has direct interference with task $\tau_{ij}$ which has direct interference with task $\tau_{kl}$; however, tasks $\tau_{ab}$ and $\tau_{kl}$ do not directly interfere with each other. Note also that for $\tau_{kl}$ to indirectly interfere with $\tau_{ab}$, the direct interference between $\tau_{ij}$ and $\tau_{kl}$ must occur before $\tau_{ij}$ interferes with $\tau_{ab}$. The indirect interference set $S_{ij}^I(\tau_{ab})$ is the set of tasks indirectly interfering with $\tau_{ab}$ through task $\tau_{ij}$. For simplicity of the formal definitions of indirect interference, we use the function *pre*$(\delta_{ij}, s_1, s_2) \rightarrow \{$*true*,*false*$\}$ to know whether stage $s_1$ precedes $s_2$ on $\delta_{ij}$.

*Definition 3:* **Indirect interference.** A communication task $\tau_{ab}$ suffers indirect interference from task $\tau_{kl}$ if and only if $(s_1 \in \delta_{ab} \cap \delta_{ij}) \wedge (s_2 \in \delta_{ij} \cap \delta_{kl}) \wedge (\delta_{ab} \cap \delta_{kl} = \emptyset) \wedge pre(\delta_{ij}, s_2, s_1) \wedge (P_{ab} < P_{ij} < P_{kl})$.

*Definition 4:* **Indirect interference set.** The set of communication tasks indirectly interfering with task $\tau_{ab}$ through task $\tau_{ij}$ is $S_{ij}^I(\tau_{ab}) = \{\tau_{kl} \mid \forall \tau_{ij} \in \Gamma_i^M, \forall \tau_{kl} \in \Gamma_k^M, (s_1 \in \delta_{ab} \cap \delta_{ij}) \wedge (s_2 \in \delta_{ij} \cap \delta_{kl}) \wedge (\delta_{ab} \cap \delta_{kl} = \emptyset) \wedge pre(\delta_{ij}, s_2, s_1) \wedge (P_{ab} < P_{ij} < P_{kl})\}$.

Tasks in the indirect interference set $S_{ij}^I(\tau_{ab})$ do not share any stage with task $\tau_{ab}$, but they must still be considered in the analysis because they can delay $\tau_{ij}$. We account for such interference by computing an indirect interference jitter term $J_{ij}^I(\tau_{ab})$ for $\tau_{ij}$ in Section IV-D. Similarly to [4], the interference jitter $J_{ij}^I(\tau_{ab})$ is then summed to the release jitter $J_{ij}^R$ to obtain the maximum jitter suffered by $\tau_{ij}$ before reaching the first stage on which it causes interference on $\tau_{ab}$.

### B. Derivation of a Response Time Estimate

We focus on deriving an upper-bound $\hat{R}$ to the response time of the task under analysis $\tau_{ab}$. Since indirect interferences are accounted for by indirect interference jitter, we only consider directly interfering tasks. Let $\{s_1, \ldots, s_M\}$ be the ordered set of stages traversed by $\tau_{ab}$ along its path $\delta_{ab}$ where $M$ is the number of stages in $\delta_{ab}$. Note that if task $\tau_{ij}$ in $S_i^D(\tau_{ab})$

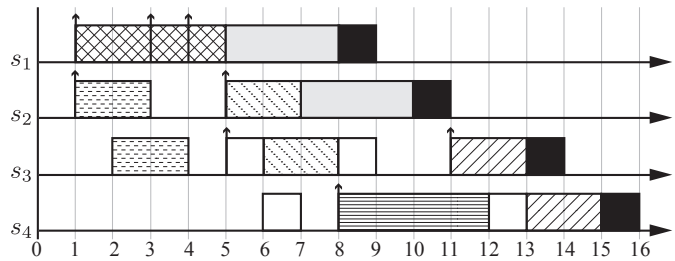



Fig. 3: An example schedule. Up arrows are release times, and the task under analysis $\tau_{abc}$ is in black.

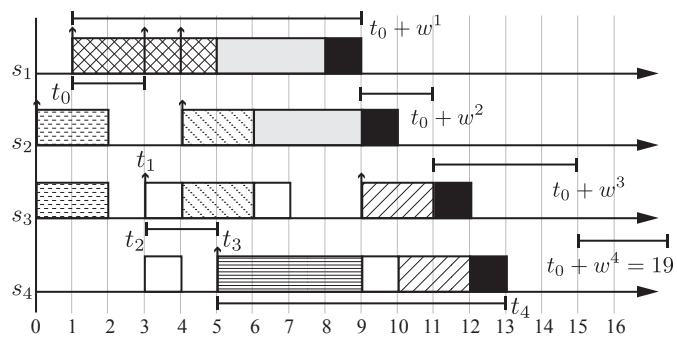



Fig. 4: An example of the stage-normalized schedule.

interferes with $\tau_{ab}$ on a non-contiguous set of stages, then $\tau_{ij}$ must be split into two or more directly interfering tasks such that each of the new tasks interferes with $\tau_{ab}$ on a contiguous set of stages.

In our derivation, we first consider the transmission of any job $\tau_{abc}$ of task $\tau_{ab}$ in any valid schedule $\Theta(t, s_k)$ on all stages of $\delta_{ab}$. We discuss how to compute an upper-bound $\hat{R}$ to the response time of $\tau_{abc}$ for that specific schedule; for clarity, we measure the response time from the activation time of $\tau_{abc}$ itself. Next, we show that the upper-bound can be maximized by modifying the pattern of release times of jobs of the tasks in $S_i^D(\tau_{ab})$, as well as the jobs of $\tau_{ab}$ itself. Finally, we show that independent of the schedule (e.g., $\Theta(t, s_k)$) and the specific job instance $c$ of $\tau_{ab}$, the proposed release time modification always yields a pattern within a finite set of *critical activation patterns*. Hence, we can derive a safe response time upper-bound for $\tau_{ab}$ by computing the maximum value of the response time upper-bound over all critical activation patterns.

We first introduce two model transformations to help us in our discussion. The transformations do not alter the transmission semantic of the model, but they simplify reasoning about the correctness of our proposed analysis. Note that based on the model in Section III, we consider that jobs of the same communication task are transmitted in FIFO order. Hence, when analyzing the job under analysis $\tau_{abc}$, we simply assume that the priority of any job $\tau_{abc'}$ that follows $\tau_{abc}$, i.e. with $c' > c$, is lower than the priority of $\tau_{abc}$. The second transformation involves the schedule $\Theta(t, s_k)$. Note that if any job $\tau_{ijp}$ is released on stage $s_k$ at time $t$, the job can not start executing on a successive stage $s_{k+l}$ (with $k + l \leq M$)

before time $t+l$. Therefore, release times can not be directly compared across different stages. To solve this issue, we define a *stage-normalized schedule* where the transmission schedule on successive stages is moved earlier in time so that release times coincide across all stages.

*Definition 5:* **Stage-normalized schedule.** Given a schedule $\Theta(t, s_k)$ over all stages in $\delta_{ab}$, the corresponding stage-normalized schedule is $\bar{\Theta}(t, s_k) = \Theta(t+k-1, s_k)$.

An example of a stage-normalized schedule is shown in Figure 4. The reported schedule is the stage-normalized version of the schedule presented in Figure 3, and it will be used as a running example throughout this section. Any datum transmitted at time $t$ on stage $s_2$ in Figure 4 is transmitted at time $t+1$ in Figure 3; any datum transmitted at time $t$ in $s_3$ in Figure 4 is transmitted at time $t+2$ in Figure 3; and so on. Also note that as a consequence of this model transformation, a datum transmitted in time slot $t$ on stage $s_k$ will now be transmitted in the same slot $t$ on stage $s_{k+1}$ if the schedule is not busy transmitting a higher priority datum. This property will significantly simplify the proofs of Lemmas 1 and 3, since it allows us to compare the busy/idle state of the schedule on two stages $s_l, s_q$ independent of the distance $q-l$ between the stages. We next formalize the concept of a busy interval on stage $s_k$, which is common to analysis of systems with fixed-job priority.

*Definition 6:* **$\mathbb{P}_{abc}$-level busy interval.** We say that $[t, t')$ is a $\mathbb{P}_{abc}$-level busy interval on stage $s_k$ in $\bar{\Theta}$ if the stage-normalized schedule continuously transmits jobs with priority greater than or equal to $\mathbb{P}_{abc}$ in slots $\langle t, \ldots, t'-1\rangle$. An interval $[t, t')$ that is not $\mathbb{P}_{abc}$-level busy is then $\mathbb{P}_{abc}$-level idle.

$\mathbb{P}_{abc}$-level busy intervals allow us to determine the interference caused by higher priority jobs on $\tau_{abc}$ in every stage $s_k$ of $\delta_{ab}$. For simplicity and since lower-priority jobs do not affect the schedule of $\tau_{abc}$ in any way, assume that all jobs in Figure 4 have higher priority than the job under analysis. Then as an example, $[0, 2)$, $[3, 7)$ and $[9, 12)$ are all maximal-length $\mathbb{P}_{abc}$-level busy intervals on $s_3$; also note that $[3, 5)$, as well as any other interval contained in a maximal-length busy interval, is a $\mathbb{P}_{abc}$-level busy interval by itself. In single-resource systems, the concept of busy interval helps the analysis because the response time of a job is necessarily bounded by the length of the unique busy interval in which it appears. However, in our situation $\tau_{abc}$ is transmitted within different $\mathbb{P}_{abc}$-level busy intervals on each stage. To effectively use the concept of busy interval, we thus define a new abstraction, called *busy chain*, which concatenates $\mathbb{P}_{abc}$-level busy intervals across all stages; as we will later prove, the busy chain is defined in such a way that each higher priority job can interfere with $\tau_{abc}$ only once.

*Definition 7:* **Busy chain.** Let $t_M$ be the time at which the job under analysis $\tau_{abc}$ finishes executing on its last stage $s_M$ in $\bar{\Theta}$, i.e., its last datum is transmitted in slot $t_M - 1$ (interval $[t_M - 1, t_M)$). For each stage $s_k$ in $\delta_{ab}$ starting from $s_M$, let $t_{k-1}$ be the earliest possible time such that $[t_{k-1}, t_k)$ is a $\mathbb{P}_{abc}$-level busy interval on $s_k$; if the stage-normalized schedule is not $\mathbb{P}_{abc}$-level busy in slot $t_k - 1$ on $s_k$, then $t_{k-1} = t_k$. Hence, $\langle [t_0, t_1), \ldots, [t_M - 1, t_M)\rangle$ is the busy chain for $\tau_{abc}$ in $\bar{\Theta}$.

It is easy to see that every job $\tau_{abc}$ admits a unique busy chain for a given stage-normalized schedule $\bar{\Theta}$. An example of a busy chain is reported in Figure 4, where $t_4 = 13, t_3 = 5, t_2 = 3, t_1 = 3, t_0 = 1$. Note that since $t_{k-1}$ is defined as the earliest possible time such that $[t_{k-1}, t_k)$ is a $\mathbb{P}_{abc}$-level busy interval on $s_k$, it follows that $\bar{\Theta}$ must be $\mathbb{P}_{abc}$-level idle in slot $t_{k-1} - 1$ (interval $[t_{k-1} - 1, t_{k-1})$); in fact, the schedule is always idle before $t_0, \ldots, t_3$ in stages $s_1, \ldots, s_4$, respectively, in Figure 4. Also note that since stage $s_2$ is $\mathbb{P}_{abc}$-level idle in $[t_2 - 1, t_2) = [2, 3)$, when analyzing $s_2$ according to the definition, we set $t_1 = t_2 = 3$.

Let $\hat{t}$ be the activation time of $\tau_{abc}$. The response time of $\tau_{abc}$ in $\bar{\Theta}$ is $t_M - \hat{t} = (t_M - t_0) - (\hat{t} - t_0)$. We can thus obtain an upper-bound on the response time of $\tau_{abc}$ by fixing $\hat{t}$ and $t_0$ and computing an upper-bound on the length of the busy chain $t_M - t_0$, similarly to how the maximum length of the busy interval is used to bound response time in [2]. The following lemma proves the key property of the busy chain for interfering higher priority jobs.

*Lemma 1:* Consider the busy chain of $\tau_{abc}$ in $\bar{\Theta}$. A given datum of any job $\tau_{ijp}$ cannot be transmitted both within the $\mathbb{P}_{abc}$-level busy interval $[t_{l-1}, t_l)$ on $s_l$ and within $[t_{q-1}, t_q)$ on $s_q$ with $q > l$.

*Proof:* Note that since $\tau_{ijp}$ is executed in the busy chain, it must hold that $\mathbb{P}_{ijp} > \mathbb{P}_{abc}$. Consider a datum of $\tau_{ijp}$ that is transmitted on $s_l$ in slot $t'_l$ with $t_{l-1} \leq t'_l < t_l$. In the stage-normalized schedule, the datum becomes ready on $s_{l+1}$ at the same time $t'_l < t_l$. By definition of a busy chain, the schedule on $s_{l+1}$ is $\mathbb{P}_{abc}$-level idle in $[t_l - 1, t_l)$. Hence, the datum must be transmitted on $s_{l+1}$ in a slot $t'_{l+1} < t_l - 1$; otherwise, it would be transmitted at $t_l - 1$ and the slot would not be $\mathbb{P}_{abc}$-level idle. If $q = l+1$, this concludes the proof; otherwise, note that $t'_{l+1} < t_l - 1$ implies $t'_{l+1} < t_{l+1}$. We can then repeat the same argument to show that on stage $s_{l+2}$, the datum is transmitted in slot $t'_{l+2} < t_{l+1} - 1$. By induction, we can then obtain $t'_q < t_{q-1} - 1$, concluding the proof. ■

Intuitively, Lemma 1 implies that every datum of an interfering job $\tau_{ijp}$ needs to be counted only once towards the length of the busy chain of $\tau_{abc}$. However, different datums of the same job can be transmitted within the busy chain on different stages. For example in Figure 4, the first datum of the task released at time 3 on $s_3$ is transmitted within $[t_2, t_3)$ on stage $s_3$, while the second datum is transmitted within $[t_3, t_4)$ on $s_4$. We use this property to compute the desired upper-bound to $t_M - t_0$ in Lemmas 2 and 3. The following two lemmas provide a way to compute the length of interfering jobs on each stage.

*Definition 8:* **Interfering job set.** $\underset{k}{S}^J$ is the set of all jobs with priority higher than or equal to $\mathbb{P}_{abc}$ that are transmitted on stage $s_k$.

*Definition 9:* **Workload.** $\bar{W}_{\underset{k}{S}^J}(t, t')$ is the sum of the transmission times of all jobs in set $\underset{k}{S}^J$ that are released in the interval $[t, t']$ in schedule $\bar{\Theta}$.

Note that $\underset{k}{S}^J$ includes $\tau_{abc}$, as well as any previous jobs of $\tau_{ab}$. Furthermore, note that by definition, the workload $\bar{W}_{\underset{k}{S}^J}(t, t')$

includes jobs that are released at time $t'$ included.

We are now ready to compute an upper bound on the length of the busy chain $t_M - t_0$. Our methodology works by induction: we first compute the maximum length of the busy chain segment $[t_0, t_1)$ on stage $s_1$ in Lemma 2 (base case). We then compute an upper bound to $[t_0, t_k)$ for each stage $s_k$ in Lemma 3 (induction step), up to stage $s_M$. The main intuition is that the busy chain is formed by a sequence of $\mathbb{P}_{abc}$-level busy intervals on each stage; hence, on each stage $s_k$, the transmission times of jobs in $\underset{k}{S}^J$ must be sufficient to continuously transmit in interval $[t_{k-1}, t_k)$. We thus bound the length of $[t_0, t_k)$ by computing the sum of transmission times of jobs that can be transmitted within continuous $\mathbb{P}_{abc}$-level busy intervals up to stage $s_k$. We will show that we can use the defined workloads $\bar{W}_{\underset{1}{S}^J}, \ldots, \bar{W}_{\underset{k}{S}^J}$ to compute such sum (note that workloads are defined based on release times of jobs, not when they are transmitted); furthermore, since Lemma 1 stipulates that a job datum cannot contribute to the busy chain on more than one stage, we will need to ensure that each job's transmission time is counted only once.

*Lemma 2:* Consider the busy chain of $\tau_{abc}$ in schedule $\bar{\Theta}$. Then for any value $\Delta$ such that:

$$\Delta = \bar{W}_{\underset{1}{S}^J}(t_0, t_0 + \Delta), \tag{1}$$

$\Delta$ is an upper-bound to $t_1 - t_0$.

*Proof:* By contradiction, assume that $t_1 > t_0 + \Delta$. Since by definition of a busy chain the schedule is $\mathbb{P}_{abc}$-level busy on $s_1$ in $[t_0, t_1)$, it follows that $[t_0, t_0+\Delta+1)$ must be a $\mathbb{P}_{abc}$-level busy interval as well. Also by definition of a busy chain, the schedule is $\mathbb{P}_{abc}$-level idle on $s_1$ in $[t_0 - 1, t_0)$. Hence, no job in $\underset{1}{S}^J$ released on $s_1$ before $t_0$ can be transmitted within the interval $[t_0, t_0 + \Delta + 1)$. Similarly, no job in $\underset{1}{S}^J$ released at or after $t_0 + \Delta + 1$ can be transmitted in $[t_0, t_0 + \Delta + 1)$. Therefore, $\bar{W}_{\underset{1}{S}^J}(t_0, t_0 + \Delta)$ is the sum of the transmission time of all jobs in $\underset{1}{S}^J$ that can be executed in $[t_0, t_0 + \Delta + 1)$. Since $\bar{W}_{\underset{1}{S}^J}(t_0, t_0 + \Delta) = \Delta$, there are not enough data units to transmit continuously in the interval $[t_0, t_0 + \Delta + 1)$, which has a duration of $\Delta + 1$ slots; this creates a contradiction. ■

Since we are interested in the tightest possible upper-bound to the response time of $\tau_{abc}$, we simply compute the minimal value $w^1$ of $\Delta$ for which Lemma 2 holds as:

$$w^1 = \min\{\Delta | \Delta = \bar{W}_{\underset{1}{S}^J}(t_0, t_0 + \Delta)\}. \tag{2}$$

As an example, when we apply Lemma 2 to the stage-normalized schedule in Figure 4, we obtain $w^1 = 8$ and thus $t_0 + w^1 = 9$, which safely over-approximates the length of the busy chain on $s_1$. In fact, it is easy to see that the value of $w^1$ is equal to the length of the longest $\mathbb{P}_{abc}$-level busy interval starting at $t_0$. Furthermore, it is also easy to see that such busy interval must have finite length as long as the sum of the utilization (e.g., $L_{ij}/T_i$) of tasks of jobs in $\underset{k}{S}^J$ is less than one; therefore, under such assumption we can always compute a valid value for $w^1$.

*Lemma 3:* For $1 \leq k \leq M$, $w^k$ is an upper-bound to $t_k - t_0$, where:

$$w^k = \min\{\Delta | \Delta = \bar{W}_{\underset{k}{S}^J}(t_0, t_0 + \Delta) + w^{k-1} - \bar{W}_{(\underset{k-1}{S}^J \bigcap \underset{k}{S}^J)}(t_0, t_0 + w^{k-1})\} \tag{3}$$

*Proof:* Note that for $k = 1$, Equation 3 reduces to Equation 2. Also, the intersection $\left(\underset{k-1}{S}^J \bigcap \underset{k}{S}^J\right)$ represents the set of jobs with priority higher than or equal to $\mathbb{P}_{abc}$ that are transmitted on $s_k$ as well as on the previous stage $s_{k-1}$.

The proof proceeds by induction. Assume that for all stages up to $s_{k-1}$, the hypothesis holds and furthermore $w^{k-1}$ includes the transmission time of all jobs in $\underset{k-1}{S}^J$ released in $[t_0, t_0 + w^{k-1}]$. By Lemma 2 and definition of $\bar{W}_{\underset{1}{S}^J}(t, t')$, this is true for $k - 1 = 1$ (base case). We need to prove that it holds for stage $s_k$ (induction step).

Similar to Lemma 2, assume by contradiction that $t_k > w^k + t_0$. We show that the maximum sum of transmission lengths of jobs transmitted in the busy chain on $s_1, \ldots, s_k$ in $[t_0, t_0 + w^k + 1)$ is $w^k$; this creates a contradiction since there are not enough data units to cover an interval of $w^k + 1$ slots. No job in $\underset{k}{S}^J$ released at or after $t_0 + w^k + 1$ can be transmitted in $[t_0, t_0 + w^k + 1)$. We will next prove that no job in $\underset{k}{S}^J$ released before $t_0$ can be executed in the busy chain on stage $s_k$ ($\mathbb{P}_{abc}$-level busy interval $[t_{k-1}, t_k)$). Finally, according to Lemma 1, any datum of a job contributing to the busy chain on stages $s_1, \ldots, s_{k-1}$ can not contribute to the busy chain on $s_k$. Therefore, we can upper-bound the sum of the transmission lengths on jobs transmitted in the busy chain in $[t_0, t_0 + w^k + 1)$ by taking the workload $\bar{W}_{\underset{k}{S}^J}(t_0, t_0 + w^k)$, summing the maximum length of the busy chain $w^{k-1}$ up to stage $s_{k-1}$, and subtracting the transmission time of jobs that are released in $\underset{k}{S}^J$ within $[t_0, t_0 + w^k + 1)$ but were already counted in $w^{k-1}$, which is $\bar{W}_{(\underset{k-1}{S}^J \bigcap \underset{k}{S}^J)}(t_0, t_0 + w^{k-1})$; this is equivalent to computing $w^k$ according to Equation 3. This concludes the induction step, since we have also shown that $w^k$ indeed includes the transmission time of all jobs in $\underset{k}{S}^J$ released in $[t_0, t_0 + w^k]$.

We still need to prove that no job in $\underset{k}{S}^J$ released before $t_0$ can be executed in the $\mathbb{P}_{abc}$-level busy interval $[t_{k-1}, t_k)$. Assume that a job $\tau_{ijp} \in \underset{k}{S}^J$ is released before $t_0$ on stage $s_l$, with $l \leq k$. Let $t'_l$ be the slot during which the last datum of $\tau_{ijp}$ is transmitted on $s_l$. Then it must be $t'_l < t_{l-1} - 1$, otherwise, $\tau_{ijp}$ would be transmitting during the $\mathbb{P}_{abc}$-idle slot $[t_{l-1} - 1, t_{l-1})$. We then use the same reasoning as in Lemma 1 to show that $t'_k < t_{k-1} - 1$, where $t'_k$ is the slot during which the last datum of $\tau_{ijp}$ is transmitted on $s_k$. ■

Figure 4 shows the values of $w^1, \ldots, w^4$ computed for the figure's schedule. Let us consider $w^2 = \bar{W}_{\underset{2}{S}^J}(t_0, t_0 + w^2) + w^1 - \bar{W}_{(\underset{1}{S}^J \bigcap \underset{2}{S}^J)}(t_0, t_0 + w^1)$. As previously discussed, $w^1$ includes the transmission times of all jobs on $s_1$ including the task under analysis $\tau_{abc}$ (in solid black). $\bar{W}_{\underset{2}{S}^J}(t_0, t_0 + w^2)$

includes the transmission times of the jobs between the time interval $[4, 10]$. Notice that the job in $[0, 2)$ is not included since it is released at time $0 < t_0 = 1$. We then subtract $\bar{W}_{(\underset{1}{S}^J \bigcap \underset{2}{S}^J)}(t_0, t_0 + w^1)$ which comprises the jobs that were included in $w^1$ but are also transmitted on $s_2$. This results in $w^2 = 10$ and $t_0 + w^2 = 11$. Similarly $t_0 + w^3 = 15$, and $t_0 + w^4 = 19$. Note that $w^1, \ldots, w^4$ significantly over-approximate the length of the busy chain; this is because the data transmitted on stage $s_1$ in $[3, 8)$ and the datum transmitted on $s_2$ and $s_3$ in $[5, 6)$ is counted in the workload despite not being part of the chain. However, as we show in the next section, the over-approximation allows us to greatly reduce the number of different job release time patterns that we need to check to find the worst-case response time of $\tau_{ab}$.

### C. Critical Activation Patterns

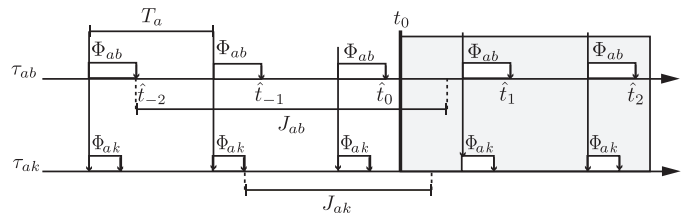
Lemma 3 gives us a way to compute an upper-bound $\bar{R}$ on the response time of $\tau_{abc}$ based on the pattern of release times of interfering jobs in $\bar{\Theta}$: (1) we first compute the bound $w^M$ for the length of the busy chain $t_M - t_0$, and (2) we then obtain $\bar{R} = w^M - (\hat{t} - t_0)$. Note that $\bar{R}$ represents the response time of $\tau_{abc}$ in the stage-normalized schedule $\bar{\Theta}$. We can compute the response time $\hat{R}$ in the original schedule $\Theta$ as $\hat{R} = \bar{R} + M - 1$. Finally, since $\hat{R}$ and $\bar{R}$ are measured from the activation time of $\tau_{abc}$, we can obtain the response time from the activation of the root vertex of application $A_a$ as $R = \hat{R} + \Phi_{ab}$. Unfortunately, this procedure is not feasible, since we would need to compute $\bar{R}$ for all jobs $\tau_{abc}$ of $\tau_{ab}$, and all possible release patterns to obtain the worst-case. To address this, we present the following lemma to show that we only need to consider a finite set of release patterns and jobs to determine the worst-case. The key idea is that we can create a worst-case pattern by releasing each interfering job as soon as possible at or after $t_0$; this maximizes the workloads computed in Lemmas 2 and 3.

*Lemma 4:* Consider applying the following rules to the release pattern of jobs in $\bar{\Theta}$:

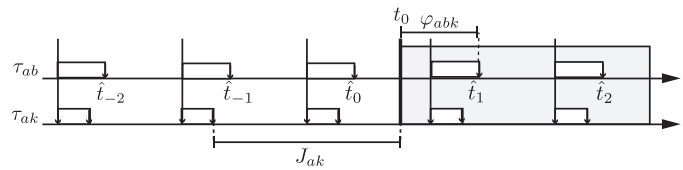
1) Every job $\tau_{ijp}$ that is activated before $t_0$ and can be released at or after $t_0$ is released at $t_0$.
2) Every job $\tau_{ijp}$ that is activated after $t_0$ is released immediately at its activation time.
3) The activation time of every application $\tau_i$ is moved earlier in time until one job of $\tau_i$ is released at time $t_0$ after suffering maximum jitter.

Then the response time bound $\bar{R}$ computed for the modified release pattern will be no less than the response time bound computed for the original pattern.

*Proof:* Consider any interval $[t_0, t_0 + \Delta]$ as in Lemmas 2 and 3. If a job was released in $[t_0, t_0 + \Delta]$ in the original pattern, then it will still be released in $[t_0, t_0 + \Delta]$ in the modified pattern. This is because Rules 1 and 2 force any job that could be released within $[t_0, t_0 + \Delta]$ to indeed be released within the interval. Furthermore, Rule 3 can not cause any job released at or after $t_0$ to be released before $t_0$. Therefore, the value of $\bar{W}_{\underset{k}{S}^J}(t_0, t_0 + \Delta)$ for any set $\underset{k}{S}^J$ computed in the modified pattern will be greater than or equal to the value computed in the original pattern. It is then easy to see that for all stages $s_k$, the computed value of $w^k$ can not decrease after applying Rules 1, 2, and 3. Since $\bar{R} = w^M - (\hat{t} - t_0)$, to conclude the proof it suffices to note that the activation time $\hat{t}$ of $\tau_{abc}$ in the modified pattern can not be larger than in the original pattern. This is because Rule 3 can move the activation time of $\tau_{abc}$ to occur earlier but not later in time. ■




(a) Original release pattern.




(b) Modified release pattern.

Fig. 5: Release patterns of jobs of application $A_a$.

Figure 5a shows an example timeline including $t_0$, the beginning of a busy chain. It also shows activation times (down arrows) $\hat{t}_{-2}, \ldots, \hat{t}_2$ for several jobs of $\tau_{ab}$, as well as jobs of another higher priority task $\tau_{ak}$ of application $A_a$. Note that the first job of $\tau_{ak}$ cannot be released at or after $t_0$. Rule 1 applies to the first three jobs of $\tau_{ab}$ and the second and third job of $\tau_{ak}$; they are activated before $t_0$ and have enough jitter to be released at or after $t_0$. These jobs are released at $t_0$. Rule 2 applies to the last two jobs of both tasks; they are activated after $t_0$. These jobs are released immediately at their activation times. Finally, Figure 5b shows the modified pattern after applying Rule 3; the activation time of $A_a$ is moved earlier in time until the second job of $\tau_{ak}$ is released at $t_0$ after suffering maximum jitter.

We call the modified pattern obtained in Lemma 4 a critical pattern. Let $\tau_{abc'}$, with $c' \leq c$, be the first job of $\tau_{ab}$ released at or after $t_0$. The number of critical patterns for $\tau_{abc}$ where $\tau_{abc'}$ is the first such job is then $(\left|S_a^D(\tau_{ab})\right| + 1) \cdot \prod_{\forall i \neq a} \left|S_i^D(\tau_{ab})\right|$. The application under analysis $A_a$ is activated at a time such that either a job of any interfering task in $S_a^D(\tau_{ab})$ or $\tau_{abc'}$ is released at $t_0$ after suffering maximum jitter, providing $\left|S_a^D(\tau_{ab})\right| + 1$ combinations. Every other application $A_i$ is activated at a time such that a job of any interfering task in $S_i^D(\tau_{ab})$ is released at $t_0$ after suffering maximum jitter, providing $\left|S_i^D(\tau_{ab})\right|$ combinations. Each of the possible critical activation patterns is characterized by a tuple $v$ of indices, one per transaction. Each index $v(i)$ identifies the task of application $A_i$ that coincides with the beginning of the busy chain. Lemma 4 represents the equivalent of Theorems 1 and 2 in Palencia and Gonzalez [2], which prove that a critical instant for the task under analysis can be built by releasing one

task of each application at the critical instant after suffering maximum jitter. Intuitively, the beginning of the busy chain $t_0$ represents the equivalent of the critical instant, except that it accounts for transmission on multiple stages and must thus include patterns that would not be valid in single-resource systems. For example, note in Figure 4 that the schedule is not $\mathbb{P}_{abc}$-level idle at time $t_0$ on stages $s_2$ and $s_3$.

Given a critical pattern, the index of the first job $\tau_{abc'}$ released at or after $t_0$ is relevant because it determines the activation time $\hat{t}$ of $\tau_{abc}$. However, the time difference $\hat{t} - t_0$ used to compute $\bar{R}$ depends only on the difference $c - c'$. In other words, the same critical release patterns apply to all jobs $\tau_{abc}$ of $\tau_{ab}$ as long as we vary the number of jobs of $\tau_{abc}$ that are released in $[t_0, t_0 + w^M]$.

### D. Indirect Interference Jitter

Given a task under analysis $\tau_{ab}$, the phase $\varphi_{ijk}$ between any task $\tau_{ij}$ and the beginning of the busy chain of a critical activation pattern created by task $\tau_{ik}$ is given by:

$$\varphi_{ijk} = T_i - (\phi_{ik} + J^R_{ik} + J^I_{ik}(\tau_{ab}) - \phi_{ij}) \bmod T_i$$

where $\phi_{ij}$ is the reduced offset of task $\tau_{ij}$ to the period 0 to $T_i$ and is given by: $\phi_{ij} = \Phi_{ij} \bmod T_i$ and $J^I_{ik}(\tau_{ab})$ is the interference jitter suffered by $\tau_{ik}$ and is given by:

$$J^I_{ik}(\tau_{ab}) = R_{ik}(\tau_{ab}) - L_{ik} - J^R_{ik} - \phi_{ik}$$

The interference jitter $J^I_{ik}(\tau_{ab})$ represents the interference suffered by flow $\tau_{ik}$ only due to flows in the indirect interference set of task $\tau_{ab}$, $S^I_{ik}(\tau_{ab})$. We use the notation $R_{ik}(\tau_{ab})$ to denote the response time of task $\tau_{ik}$ only due to interference from higher priority tasks in the set $S^I_{ik}(\tau_{ab})$. As an example, Figure 5b shows the phase $\varphi_{abv(a)} = \varphi_{abk}$ between $\tau_{ab}$ and the beginning of the busy chain created by $\tau_{ak}$, assuming that $J_{ak} = J^R_{ak} + J^I_{ak}(\tau_{ab})$. Note that for the first job of $\tau_{ab}$ activated after $t_0$, $\varphi_{abk}$ is exactly equal to the length of the interval $\hat{t} - t_0$ used in Section IV-C.

### E. Exponential Analysis

Now we are ready to present the formulation for the exponential response time analysis. Let us use the term critical instant for the beginning of the busy chain, $t_0$, of a critical activation pattern. We introduce a numbering scheme to track the number of job instances that we need to consider in a busy chain. We use $p = -1$ to refer to the instance activated in the interval $[t_0 - 2*T_a, t_0 - T_a)$, $p = 0$ in the interval $[t_0 - T_a, t_0)$, $p = 1$ in the interval $[t_0, t_0 + T_a)$, and so on. Note that the numbering scheme corresponds with the indices of $\hat{t}_p$ in Figure 5; furthermore, we have $\hat{t}_p - t_0 = \varphi_{abv(a)} + (p-1)*T_a$. The first job instance that we consider is one with the least index that has enough jitter to be part of the busy chain. Hence, the first job instance $p^v_{0,ab} = -\lfloor \frac{J^R_{ab} + \varphi_{abv(a)}}{T_a} \rfloor + 1$.

*Lemma 5:* The worst-case contribution of an application $A_i$ to the busy chain for $\tau_{ab}$ on stage $s_l$ when the activation of task $\tau_{ik}$ coincides with the critical instant is given by:

$$W_{ik}(\tau_{ab}, l, t) = \sum_{\forall j \in \underset{l}{S}^D_i(\tau_{ab})} \Big( \Big\lfloor \frac{J^R_{ij} + J^I_{ij}(\tau_{ab}) + \varphi_{ijk}}{T_i} \Big\rfloor + \Big\lfloor \frac{t - \varphi_{ijk}}{T_i} \Big\rfloor + 1 \Big) * L_{ij}$$

*Proof:* By definition, a workload $\bar{W}_{\underset{l}{S}^J}(t, t')$ includes the transmission times of all jobs in the set $\underset{l}{S}^J$ that are released in the interval $[t, t']$. The set $\underset{l}{S}^J$ includes all jobs of higher priority tasks of application $A_i$, $\underset{l}{S}^D_i(\tau_{ab})$, that contribute to the workload on stage $s_l$. Consider an arbitrary higher priority task of application $A_i$, $\tau_{ij}$. According to Rule 1 of Lemma 4, all job instances of $\tau_{ij}$ that are activated before the critical instant and have enough jitter that allows them to contribute to the workload, are released at the critical instant. The first term of the summation accounts for these job instances, this is similar to the derivation in [2]. The second term simply applies Rule 2 of Lemma 4. The job instances belonging to Rule 2 are a series of periodic activations starting $\varphi_{ijk}$ time units after the critical instant. Note, however, that by definition of a workload, we need to account for activations released at time $t'$ (end of the interval) included. Hence, the 1 added to the second term to account for these activations. ■

We further explain Lemma 5 using Figure 5b applied to task $\tau_{ak}$, where $v(a) = k$. The first term $\lfloor \frac{J^R_{ak} + J^I_{ak}(\tau_{ab}) + \varphi_{akk}}{T_a} \rfloor$ accounts for the second and third jobs that are activated before the critical instant. The second term $\lfloor \frac{t - \varphi_{akk}}{T_a} \rfloor + 1$ accounts for the two jobs that are activated after the critical instant. Note that according to the definition of the workload, if the nominator $t - \varphi_{akk}$ is such that it exactly matches the period, then we will still consider the last job to be part of the worst-case contribution $W_{ak}(\tau_{ab}, l, t)$.

For clarity of presentation, we define $W'_{ik}(\tau_{ab}, l, t)$ as the worst-case contribution of an application $A_i$ on the response time of task $\tau_{ab}$ on stage $s_{l-1}$ solely due to tasks that are common on stages $s_{l-1}$ and $s_l$, i.e., in the set $\underset{l-1}{S}^D_i(\tau_{ab}) \cap \underset{l}{S}^D_i(\tau_{ab})$. We also use $\underset{l}{p}^v_{B,ab}$ to denote the largest-numbered job instance in the interval $w^l$.

*Lemma 6:* For each activation pattern $v$, the worst-case length of the busy chain for each job $p$ of task $\tau_{ab}$ up to stage $s_l$ is determined by:

$$\underset{l}{w}^v_{ab}(p) = \underset{l-1}{w}^v_{ab}(p') + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, \underset{l}{w}^v_{ab}(p)) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab}(p')) + (p - p^v_{0,ab} + 1) * L_{ab}$$

where $p' = p$ if $p \leq \underset{l-1}{p}^v_{B,ab}$, else $p' = \underset{l-1}{p}^v_{B,ab}$.

*Proof:* This proof directly descends from Lemma 3. In the above equation, if we remove the restriction to a specific job instance $p$, then we compute $\underset{l}{w}^v_{ab}$ which is the length of the busy chain up to stage $s_l$ where $c = \underset{l}{p}^v_{B,ab}$. Thus, computing $w^l$ as introduced in Lemma 3. The only difference is that

$\underset{l}{w}^v_{ab}$ separates interference from higher priority tasks and jobs from the same task into different terms. In order to compute the $\underset{l}{w}^v_{ab}(p)$ for a specific job instance then we only consider interference from higher priority tasks while excluding any job instances that are activated after $p$. Similar to Lemma 3, and using Lemma 5, to compute $\underset{l}{w}^v_{ab}(p)$ we use $\underset{l-1}{w}^v_{ab}(p')$ plus any interference on stage $s_l$ while subtracting interferences that are common on stages $s_{l-1}$ and $s_l$. We restrict instances from the task $\tau_{ab}$ only up till $p$. Note that we use $p'$ instead of $p$ in $\underset{l-1}{w}^v_{ab}(p')$. This is due to the fact that a certain $p$ might only exist on stage $s_l$ but not $s_{l-1}$, making $\underset{l-1}{w}^v_{ab}(p)$ undefined. In such case, the $w$ that can be considered from the previous stage is only up till the maximum $p$ existing on it, i.e., $\underset{l-1}{p}^v_{B,ab}$. ■

Note that for stage $s_1$, $\underset{1}{w}^v_{ab}(p)$ is equal to $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, 1, \underset{1}{w}^v_{ab}(p)) + (p - p^v_{0,ab} + 1) * L_{ab}$.

To obtain $\underset{l}{p}^v_{B,ab}$ for any stage $s_l$, we simply replace $\underset{l}{w}^v_{ab}(p)$ and $\underset{l-1}{w}^v_{ab}(p')$ by $\underset{l}{w}^v_{ab}$ and $\underset{l-1}{w}^v_{ab}$, i.e., we compute a busy interval instead of a response time of a specific job instance. We also replace $p$ by $(\lfloor \frac{\underset{l}{w}^v_{ab} - \varphi_{abv(a)}}{T_a} \rfloor - \lfloor \frac{\underset{l-1}{w}^v_{ab} - \varphi_{abv(a)}}{T_a} \rfloor)$. This simply accounts for new job instances of $\tau_{ab}$ that exist on stage $s_l$ but not $s_{l-1}$.

*Theorem 1:* The worst-case response time of task $\tau_{ab}$ is obtained by:

$$R_{ab} = \max_{\forall v} \Big( \max_{p = p^v_{0,ab} \cdots \underset{M}{p}^v_{B,ab}} (\underset{M}{R}^v_{ab}(p)) \Big) + M - 1$$

where $\underset{M}{R}^v_{ab}(p) = \underset{M}{w}^v_{ab}(p) - \varphi_{abv(a)} - (p-1) * T_a + \Phi_{ab}$.

*Proof:* Using Lemma 6, we can find the worst-case length of the busy chain for each job $p$ on stages along the path $\delta_{ab}$ of $\tau_{ab}$ for a particular activation pattern $v$. We also showed that the response time for a job of $\tau_{ab}$ is $\bar{R} = w^M - (\hat{t} - t_0)$, i.e, $\bar{R} = \underset{M}{w}^v_{ab}(p) - \varphi_{abv(a)} - (p-1) * T_a$ for job $p$. To obtain the response time $\underset{M}{R}^v_{ab}(p)$ measured from the activation of application $A_a$, rather than the activation of $\tau_{ab}$, we also need to add the offset $\Phi_{ab}$. Next, we compute the maximum worst-case response time across all job instances, which gives us the worst-case response time of $\tau_{ab}$ in $v$. From Lemma 4, we consider the maximum worst-case response time across all activation patterns to obtain the WCRT of task $\tau_{ab}$, $R_{ab}$. ■

### F. Polynomial Analysis

The difficulty with the exponential analysis is that it is exponential in the number of critical activation patterns. Hence, we derive an upper-bound on the interference caused by an application $A_i$ as the maximum interference caused by considering each task in $A_i$ to coincide with the critical instant. We use the notation $W^*_i(\tau_{ab}, l, t) = \max_{\forall k \in \underset{l}{S}^D_i(\tau_{ab})} W_{ik}(\tau_{ab}, l, t)$ to compute this upper-bound on a specific stage $s_l$. We use this approximation only for higher priority applications and not application $A_a$ to which the task under analysis $\tau_{ab}$ belongs. The number of critical activation patterns that we must consider is thus reduced to $\left|S^D_i(\tau_{ab})\right| + 1$. In what follows, we derive a polynomial WCRTA for our pipelined communication model. For clarity of presentation, we use $W''^*_i(\tau_{ab}, l, t) = \max_{\forall k \in \underset{l}{S}^D_i(\tau_{ab})} W''_{ik}(\tau_{ab}, l, t)$ where $W''_{ik}(\tau_{ab}, l, t)$ is the worst-case contribution of an application $A_i$ to the busy chain for task $\tau_{ab}$ on stage $s_{l-1}$ solely due to tasks that are on stage $s_{l-1}$ but not $s_l$, i.e., in the set $\underset{l-1}{S}^D_i(\tau_{ab}) \backslash \underset{l}{S}^D_i(\tau_{ab})$.

*Lemma 7:* For a critical instant created with task $\tau_{ac}$, the worst-case length of the busy chain for each job $p$ of task $\tau_{ab}$ up to stage $s_l$ is determined by:

$$\begin{aligned}\underset{l}{w}_{abc}(p) = & \sum_{s=2\ldots l} \sum_{\forall i \neq a} W''^*_i(\tau_{ab}, s, \underset{s-1}{w}_{abc}(p')) \\ & + \sum_{\forall i} W^*_i(\tau_{ab}, l, \underset{l}{w}_{abc}(p)) + \sum_{s=2\ldots l} W''_{ac}(\tau_{ab}, s, \underset{s-1}{w}_{abc}(p')) \\ & + W_{ac}(\tau_{ab}, l, \underset{l}{w}_{abc}(p)) + (p - p^v_{0,ab} + 1) * L_{ab}\end{aligned}$$

*Proof:* We first transform the interference from Lemma 6 into a more convenient form for the discussion. Consider using Lemma 6 to compute a busy interval (through dropping $p$ as shown earlier). Let us focus on the interference from higher priority tasks and disregard jobs from the same task (they are not affected by the approximation). The interference is accounted for by the terms $\underset{l-1}{w}^v_{ab} + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, \underset{l}{w}^v_{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab})$ of Lemma 6. The term $\underset{l-1}{w}^v_{ab}$ can be expanded into $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, l-1, \underset{l-1}{w}^v_{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l-1, \underset{l-2}{w}^v_{ab})$. We expand all terms until reaching the first stage $s_1$. Hence, we get:

$$\begin{aligned}& \sum_{\forall i} W_{iv(i)}(\tau_{ab}, 1, \underset{1}{w}^v_{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, 2, \underset{2}{w}^v_{ab}) \\ & - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, 2, \underset{1}{w}^v_{ab}) + \ldots \\ & + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l-1, \underset{l-1}{w}^v_{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l-1, \underset{l-2}{w}^v_{ab}) \\ & + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, \underset{l}{w}^v_{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab})\end{aligned}$$

Recall that $\sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab})$ is the interference on stage $s_{l-1}$ from tasks in the set $\underset{l-1}{S}^D_i(\tau_{ab}) \cap \underset{l}{S}^D_i(\tau_{ab})$. Hence the terms, $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, l-1, \underset{l-1}{w}^v_{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab})$ can be rewritten as $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab})$ which is the interference due to tasks in the set $\underset{l-1}{S}^D_i(\tau_{ab}) \backslash \underset{l}{S}^D_i(\tau_{ab})$. This is intuitive since the first term is interference from the set $\underset{l-1}{S}^D_i(\tau_{ab})$ and the second term is the interference from the set $\underset{l-1}{S}^D_i(\tau_{ab}) \cap \underset{l}{S}^D_i(\tau_{ab})$. So their difference yields the interference in the set $\underset{l-1}{S}^D_i(\tau_{ab}) \backslash \underset{l}{S}^D_i(\tau_{ab})$. Therefore, we can write the interference in Lemma 6 in the form: $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, 2, \underset{1}{w}^v_{ab})$ + … + $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, l, \underset{l-1}{w}^v_{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, \underset{l}{w}^v_{ab})$. Or in a more compact form $\sum_{s=2\ldots l} \sum_{\forall i} W''_{iv(i)}(\tau_{ab}, s, \underset{s-1}{w}^v_{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, \underset{l}{w}^v_{ab})$.

Next, we consider the first stage on path $\delta_{ab}$ of $\tau_{ab}$. The interference on stage $s_1$ is equal to $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, 1, \underset{1}{w}^v_{ab})$. Using the approximation introduced earlier, the interference will be equal to $\sum_{\forall i} W^*_i(\tau_{ab}, 1, \underset{1}{w}^v_{ab})$. Since the approximation considers the maximum interference that can be achieved by $A_i$ through considering each task to coincide with the critical instant, then doing the summation over all higher priority applications yields an interference that is greater than or equal to considering any individual activation pattern, i.e., $W^*_i(\tau_{ab}, 1, \underset{1}{w}^v_{ab})$ is an upper-bound of $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, 1, \underset{1}{w}^v_{ab})$. This is similar to the derivation in [11].

Let us consider the second stage $s_2$. The interference on this stage is equal to $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, 2, \underset{1}{w}^v_{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, 2, \underset{2}{w}^v_{ab})$. This is the sum of interferences from tasks that are only on stage 1 and tasks that exist on stage 2. Now consider the approximation $\sum_{\forall i} W''^*_i(\tau_{ab}, 2, \underset{1}{w}^v_{ab}) + \sum_{\forall i} W^*_i(\tau_{ab}, 2, \underset{2}{w}^v_{ab})$. Since $W^*_i(\tau_{ab}, 2, \underset{2}{w}_{abc}) \geq W_{iv(i)}(\tau_{ab}, 2, \underset{2}{w}^v_{ab})$ and the same holds for the interference that occurs only on stage 1, $W''^*_i(\tau_{ab}, 2, \underset{1}{w}^v_{ab}) \geq W''_{iv(i)}(\tau_{ab}, 2, \underset{1}{w}^v_{ab})$. Hence, the approximation on stage $s_2$ yields an upper-bound to the interference on that stage. Similarly this can be extended to all stages until stage $M$.

Lastly, in this Lemma, we only apply the approximation to higher priority applications, i.e., $\forall i \neq a$. Hence, we compute an upper-bound for the interference from all higher priority applications. To find the polynomial worst-case response time for task $\tau_{ab}$, we need to consider all possible critical instants from application $A_a$. ■

*Theorem 2:* The worst-case response time of task $\tau_{ab}$ is obtained by:

$$R_{ab} = \max_{\forall c \in S^D_i(\tau_{ab}) \cup b} \left( \max_{p = p^v_{0,ab} \cdots \underset{M}{p}^v_{B,ab}} (\underset{M}{R}_{abc}(p)) \right) + M - 1$$

where $\underset{M}{R}_{abc}(p) = \underset{M}{w}_{abc}(p) - \varphi_{abv(a)} - (p-1) * T_a + \Phi_{ab}$.

*Proof:* The proof is similar to the proof of Theorem 1. ■

## V. Experimental Evaluation

For the experimental evaluation of the proposed WCRTA, we present an application of a priority-aware NoC as presented by Shi and Burns [4], [5], [6]. This NoC supports wormhole switching with flit-level preemption. Details of the NoC architecture are available in [4]. In particular, we propose two instances of NoCs with sizes $4\times4$ and $8\times8$ for the deployment platform. Each node in the NoC is a PR in the processing and communication resource model, and each link between two nodes represents a stage in the pipelined communication resources. A computation task executes on the node, and the communication task transmits data across multiple links to its destination node. These links of the NoC correspond to the different stages of the CR. We experiment with exponential and polynomial versions of the offset-based flow-level analysis (OFLA) [8], and Palencia and Gonzalez's [2] (PAL) analyses,

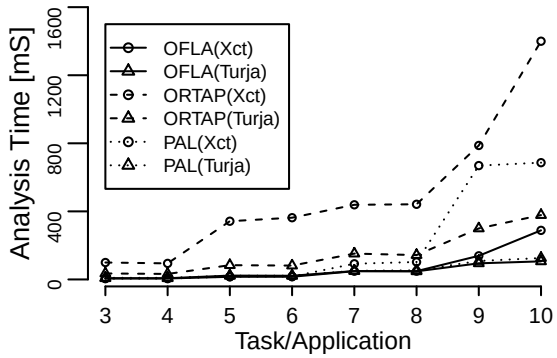


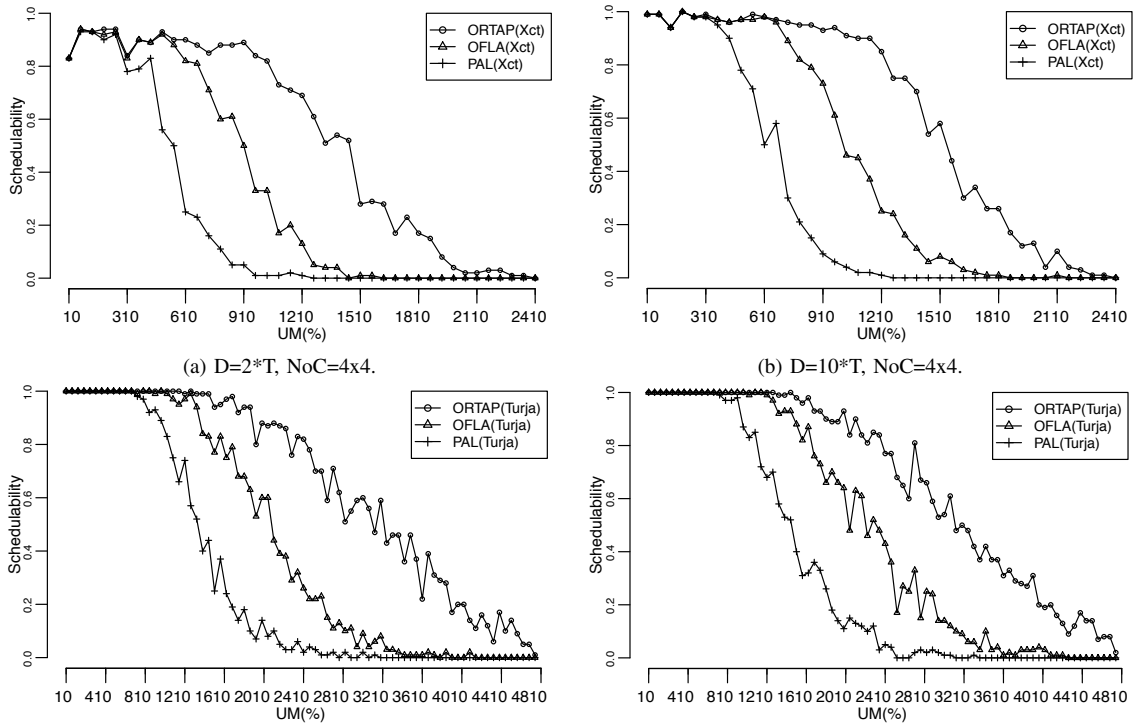

Fig. 6: Run-time comparison of various WCRTA techniques.

and the proposed analysis (ORTAP). We also add the optimization by Turja and Maki [3] to all polynomial versions. We randomly generate DAGs to represent arbitrary applications with a specified number of tasks.

We setup the experiments with the following parameters. We use 10 applications per test. The number of tasks per application is varied in the range (3,10). The application period is randomly chosen in the range (1.000,1.000.000) and the deadline is chosen as a coefficient (e.g. 10x) of the period. The applications are prioritized using rate-monotonic priority assignment, and an arbitrary priority assignment scheme is chosen for priority of tasks within each application. The application release jitter is set to zero (i.e. $J^R = 0$). Task offsets and jitters are calculated based on the methods presented in Section III-C. The communication utilization (UM) is equally divided between applications ranging from 10% to 4800% in steps of 60 and the PR utilization is set to 500%. Random application mapping is used, and for the communication tasks the routes are selected by a shortest path algorithm. For each configuration, 100 random test cases are executed.

Figure 6 displays the run-time of the six WCRTA techniques. This includes the exponential and polynomial versions for ORTAP, OFLA, and PAL. The results show that the run-time of the exponential versions grow exponentially as we increase the number of tasks per application when compared to the polynomial. It also shows that the run-time for the exponential ORTAP is larger than others with the exception of exponential PAL. Exponential ORTAP performs the WCRTA for an exponential number of activation patterns on multiple stages, thus the large run-time. Exponential PAL considers indirect interference as direct interference. This increases the number of activation patterns that have to be considered which is exponential in an exponential analysis. Thus, leading to a higher execution time even compared to exponential ORTAP.

Figure 7 shows the average schedulability of application for the various WCRTA techniques. For each figure, we have 10 tasks per application, and for each NoC size we use the deadline of each application to be twice its period, and one where the deadline is ten times its period. Figure 7a shows that the schedulability of the exponential version of ORTAP is higher compared to both the exponential versions of OFLA and PAL. Furthermore, as we increase the utilization of the




(a) D=2*T, NoC=4x4.

(b) D=10*T, NoC=4x4.

(c) D=2*T, NoC=8x8.

(d) D=10*T, NoC=8x8.

Fig. 7: Schedulability of application sets with 10 tasks per application.

NoC, we observe that ORTAP has a higher schedulability than the other analysis techniques. This is because ORTAP is able to better analyze workloads with a large amount of interference per stage. Since such interferences do not exist when the utilization is low, all three techniques do equally well. This holds for all graphs in Figure 7. From Figure 7b, we make the same observations with the period set to ten times that of the period. Notice that increasing the deadline results to higher schedulability for all analyses simply because the deadline is larger. Figure 7c and 7d shows the schedulability of the polynomial versions of the analysis. Once again, the polynomial version of ORTAP outperforms OFLA and PAL in terms of schedulability.

## VI. Conclusion

We present a WCRTA analysis for a pipelined communication resource model. A concrete application of this model is in estimating worst-case latencies across communication interconnects such as a priority-aware NoC. In developing this WCRTA technique, we construct an exponential analysis, and its corresponding polynomial analysis. We provide proofs of correctness to ensure that the analysis does indeed provide the upper-bounds. To evaluate the analysis, we create two instances of a NoC, and deploy a large suite of synthetic benchmarks. These synthetic benchmarks are essential and necessary to stress the analysis technique. We compare our WCRTA against prior works, and our results show that the schedulability of ORTAP is 48.3% and 66.7% higher than the schedulability of OFLA and PAL for the two NoC sizes.

## References


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

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

[3] 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.

[4] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *IEEE International Symposium on Networks-on-Chip*, 2008, pp. 161–170.

[5] ——, "Schedulability analysis and task mapping for real-time on-chip communication," *Real-Time Systems*, vol. 46, pp. 360–385, 2010.

[6] ——, "Priority assignment for real-time wormhole communication in on-chip networks," in *Real-Time Systems Symposium*, 2008, pp. 421 – 430.

[7] P. Jayachandran and T. Abdelzaher, "A delay composition theorem for real-time pipelines," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, ser. ECRTS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 29–38.

[8] S. Gholamian, H. Kashif, H. D. Patel, R. Pellizzoni, and S. Fischmeister, "HolisticNoC: A NoC-Aware Holistic Analysis for Distributing Hard Real-time Systems on CMPs," Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Tech. Rep. CAESR-TR-2012-05, September 2013.

[9] 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 ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, jul 2012, p. 499504.

[10] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test*, vol. 22, no. 5, pp. 414–421, Sep. 2005. [Online]. Available: http://dx.doi.org/10.1109/MDT.2005.99

[11] K. Tindell, "Adding time-offsets to schedulability analysis," Dept. of Computer Science, University of York, Tech. Rep., January 1994.