

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

Hany Kashif, Sina Gholamian, Rodolfo Pellizzoni, Hiren Patel and Sebastian Fischmeister

Abstract—This work addresses the challenge of computing end-to-end 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 takes into account the dynamic offsets and jitters between application tasks. The employed pipelined communication resource 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 article, we first present a holistic analysis for computing the dynamic offsets and jitters of application tasks. We present an offset-based flow-level analysis (OFLA) for calculating the WCRTs of applications deployed on a multi-processor system. We also present ORTAP, a WCRTA that incorporates the pipelined nature of the communication in the resource model. For both OFLA and ORTAP, we present an exponential analysis, and a polynomial analysis, and prove their correctness. We compare the proposed analyses against prior analysis techniques. We perform our experimental evaluation on two instances of 4×4 and 8×8 priority-aware NoCs with 512,000 synthetic benchmarks. OFLA shows a 38.1% improvement in schedulability for the two NoC sizes over prior work. ORTAP shows a 48.3% and 66.7% improvement in schedulability for the two NoC sizes over OFLA and prior work.

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



1 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. Hence, recent research by Kashif et al. [4] focuses on deriving tight worst-case latencies on multi-processor platforms by considering the pipelined nature of the communication resources.

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. We extend the flow-level analysis (FLA) proposed by Shi and Burns [5], and introduce dynamic offsets for dependencies between computation and communication tasks modelled using a DAG. We present the theory for OFLA (Offset-based FLA) which includes two variants of the analysis. The first is an exponential analysis, and the second is polynomial. In general, the exponential analysis is intractable; however, the polynomial analysis is tractable. We

• H. Kashif, S. Gholamian, R. Pellizzoni, H. Patel, and S. Fischmeister 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, rpellizz@uwaterloo.ca, hdpatel@uwaterloo.ca, sfischme@uwaterloo.ca

also propose ORTAP, a WCRTA technique that extends the stage-level analysis (SLA) proposed by Kashif et al. [4]. SLA is a worst-case communication latency analysis for computing worst-case latencies of communication tasks in a pipelined communication resource model. ORTAP extends SLA to use dynamic offsets to accurately model the dependencies between the computation and communication tasks of an application. We present the theory behind ORTAP, which also includes two variants of the analysis; exponential and polynomial. Our experimental evaluation of these analyses uses a large suite of synthetic benchmarks varying the utilization on priority-aware NoCs (as an application of a pipelined communication resource model) [5], [6], [7]. We contend that using synthetic benchmarks for the NoC application is the correct method for evaluating the strengths and weaknesses of the proposed WCRTA techniques. 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 techniques against the work by Palencia and Gonzalez [2]. Every approximated analysis technique uses the methodology proposed by Maki-Turja and Nolin [3]. We show application schedulability, and the execution times of the analysis for both the exponential and polynomial variants of the proposed WCRTA techniques. The results show that the schedulability of OFLA is 38.1% higher than that of Palencia. The results also indicate that the schedulability of ORTAP is 48.3% and 66.7% higher than the schedulability of OFLA and Palencia, respectively.

2 MAIN CONTRIBUTIONS

In this article, we have the following contributions:

- A holistic analysis for computing the dynamic offsets and jitters for application tasks modelled using a DAG.
- Exponential and polynomial offset-based flow level analysis (OFLA) for computing application WCRTs.
- Exponential and polynomial offset-based response-time analysis for pipelined communication resources (ORTAP).
- Apply the approximations proposed by Maki-Turja and Nolin [3] to the polynomial analyses.
- Compare the performance of OFLA, ORTAP, and Palencia [2] using a large set of synthetic benchmarks.

3 ORGANIZATION

This article is organized as follows: Section 4 discusses related work. Section 5 presents the system model. Section 6 presents a holistic analysis for the computation of offsets and jitters of computation and communication tasks. Sections 7 and 8 present the theory for OFLA and ORTAP, respectively. Section 9 presents and discusses the experimental results. Section 10 concludes the article.

4 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 the number of tasks, Palencia and Gonzalez [2] present an approximated version of it that is computable in polynomial time. Later on, Maki-Turja and Nolin [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 [8] 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.

This work 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 [5], [6], [7]. Note that our work focuses on the priority-aware NoC proposed by Shi and Burns [5], [6], [7], [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. This work also presents ORTAP, an offset-based WCRTA that accounts for the pipelined communication in pipelined communication resource models. ORTAP is based on the worst-case latency analysis presented by Kashif et al. [4]. OFLA is significantly different from ORTAP. 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. ORTAP, on the other hand, analyzes interferences at each stage of the pipeline. Although, OFLA does not incorporate an accurate representation of modern interconnects, it is a simpler analysis with a shorter analysis time. In this article, we compare ORTAP, OFLA, and Palencia [2].

5 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.

5.1 Resource Model

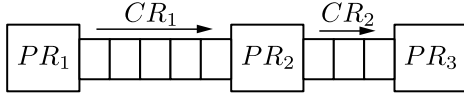


Fig. 1: Illustrative example of processing and communication resource model.

We use the same processing and communication resource model that was introduced in [4]. This is 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 [11].

One possible architectural implementation of our proposed interconnect is a worm-hole switched priority-aware network-on-chip with flit-level preemption [5]. 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.

5.2 Task Model

We model a real-time system as a set of n applications $\mathcal{A} := \{A_1, A_2, \dots, A_n\}$ 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 $G_{A_i} = (\Gamma_i^C, \Gamma_i^M)$ consisting of a set of nodes Γ_i^C that represent computation tasks

and a set of edges Γ_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) $v_{s_{ik}}$, 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) δ_{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 τ_{ik} takes to transmit data on a single communication resource when it does not suffer interferences from any other tasks. The worst-case transmission latency of task τ_{ik} along its path is $C_{ik} = L_{ik} + M - 1$ where M is the number of stages on the path of communication task τ_{ik} . We use the notation τ_{ikc} to refer to the c -th instance (job) of task τ_{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 τ_{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 complete 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 [12].

Figures 2a and 2b present an illustrative example of a DAG as the task graph for an application A_1 and its mapping onto a mesh resource model. Ellipses identify computation tasks, and arrows represent communication tasks. For example, computation task τ_{11} has a WCET of 5 time units, and communication task τ_{13} has worst-case latency of 2 time units. Figure 2b shows the mapping of A_1 onto a 2×3

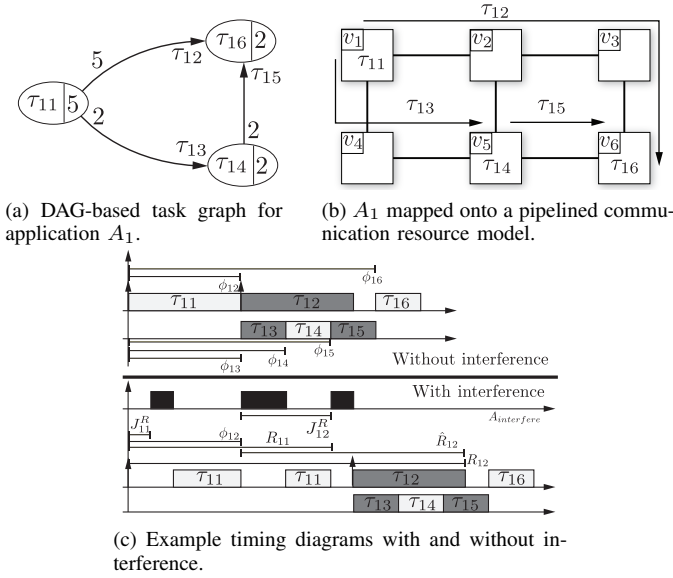


Fig. 2: Illustrative example application A_1 : Task graph and its mapping.

mesh. As an example, τ_{11} and τ_{16} are mapped onto v_1 and v_6 , with a communication task τ_{12} that traverses the path $\langle (v_1, v_2), (v_2, v_3), (v_3, v_6) \rangle$.

5.3 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 τ_{ik} in the application is activated after a specific time interval from the activation of the root vertex. We call this time interval, offset Φ_{ik} , which is the best-case release time of task τ_{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 Φ_{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. In Section 6, we

Algorithm 1: Holistic analysis.

Input: Application set $\mathcal{A}$, Mapping of tasks from $\mathcal{A}$ to resources, and Communication routes δ .
Output: Offsets, jitters, WCRTs for each task, and end-to-end WCRT for all applications.

```

forall the  $A_i \in \mathcal{A}$  do
  forall the  $\tau_{ik} \in T_C \cup T_M, k > 1$  do
    Offsets( $\tau_{ik}$ )  $\leftarrow$  ComputeOffsets( $\tau_{ik}$ )
    Jitters( $\tau_{ik}$ )  $\leftarrow$  0
  end
  Offsets( $\tau_{i1}$ )  $\leftarrow$  0
  Jitters( $\tau_{i1}$ )  $\leftarrow$   $J_i^R$ 
end
while  $\neg$ fixpoint(Jitters, ResponseTime) and ResponseTime  $< D$  do
  forall the  $A_i \in \mathcal{A}$  do
    forall the  $\tau_{ik} \in T_C \cup T_M$  do
      ResponseTime( $\tau_{ik}$ )  $\leftarrow$  ComputeWCRT( $\tau_{ik}$ )
      Jitters( $\tau_{ik}$ )  $\leftarrow$  ComputeJitters( $\tau_{ik}$ , ResponseTime(pre( $\tau_{ik}$ )))
    end
  end
forall the  $A_i \in \mathcal{A}$  do
  AppResponseTime( $A_i$ )  $\leftarrow$  ComputeAppWCRT( $A_i$ )
  if AppResponseTime( $A_i$ )  $\geq D_i$  then
    | Unschedulable( $A_i$ )  $\leftarrow$  TRUE
  end
end

```

introduce a holistic analysis [1] that iteratively computes the jitters followed by the WCRT of each task.

Figure 2c shows a schedule for the task graph in Figure 2a to illustrate offsets, jitters, and response times. Up arrows denote release times. The timing diagrams show execution sequences without and with interference where $A_{interfere}$ is some higher priority interfering application. Notice in Figure 2c (without interference), that τ_{12} can at best start after the WCET of task τ_{11} because it is dependent on τ_{11} completing its execution. Hence, the offset ϕ_{12} of task τ_{12} is its best case release time which is equal to 5 time units. Similarly, jitters are used to model worst-case temporal dependencies among tasks. In the case with interference (lower diagram in Figure 2c), τ_{11} suffers a release jitter of 1 time unit. Task τ_{11} releases at time 1; however, it could be released at any time from 0 up to and including its maximum release jitter, which is the application's release jitter $J_{11}^R = J_1^R$. Although task τ_{11} is released at time 1, it experiences interference from a task in $A_{interfere}$ delaying its execution to time 2. Task τ_{11} suffers another interference from another task of $A_{interfere}$ at time 5 causing τ_{11} to delay its end time to 9. The worst-case response time R_{11} of τ_{11} is 9. Task τ_{12} releases at time 9, which is the end time for τ_{11} . The difference between the release time of τ_{12} and its Φ_{12} provides the release jitter J_{12}^R . The release time of τ_{12} coincides with some other interfering task of $A_{interfere}$ such that it gets delayed even further. The WCRT of τ_{12} is denoted by R_{12} , which is 15. Note that we denote $\hat{R}_{12}$ as the WCRT from the task's activation. For τ_{12} , this is 9.

6 HOLISTIC ANALYSIS

Algorithm 1 presents the key steps in our holistic analysis. We assume that the application set $\mathcal{A}$, the mapping of tasks to the resources, and the communication routes are given. The algorithm starts by computing initial offsets and jitters for each task. The first task τ_{i1} of each application has a release jitter equal to the application's release jitter ($J_{i1}^R = J_i^R$), and since it is the root vertex of the DAG, its offset is zero ($\phi_{i1} = 0$). All other tasks are assigned a jitter of zero and an offset derived using `ComputeOffsets` (see Section 6.1). Then, the algorithm computes the WCRT of every task in every application using `ComputeWCRT`. For communication tasks, `ComputeWCRT` uses either OFLA (Section 7) or ORTAP (Section 8). For computation tasks, `ComputeWCRT` employs traditional offset-based WCRT analysis for tasks on uniprocessors [2]. Please see [2] for further details. Note that we restrict the set of higher priority tasks that interfere with a task under analysis τ_{ab} to those tasks that are mapped to the same PE as τ_{ab} . The task jitter is derived using `ComputeJitters` (see Section 6.2), which uses the WCRT of the precedent tasks (denoted by pre). The algorithm iterates until it reaches a fixpoint. This is when the jitters and WCRTs do not change, or a task misses the application deadline. Note that if no deadline is missed, then the algorithm is guaranteed to converge to a fixpoint because the WCRT analysis is monotonic in the release jitters [2], i.e., increasing the jitters can not cause the computed WCRT for any task to decrease. Finally, the WCRT of the application is obtained using `ComputeAppWCRT` as the maximum WCRT among all exit tasks of the application. An unschedulable application is one whose WCRT is larger than the deadline.

6.1 ComputeOffsets: Computing Offsets

Unlike prior works [2], [3] that only support a linear temporal dependency between tasks of a transaction, we support DAGs, which allow for parallel dependencies between tasks. Consequently, we compute offsets for tasks as follows.

$$\phi_{ik} = \max_{\forall \rho_{ik} \in \Pi_{ik}} \sum_{\forall \tau_{ij} \in \rho_{ik}} C_{ij} \quad (1)$$

where ϕ_{ik} is the offset for task τ_{ik} , Π_{ik} is the set of all paths in the application DAG starting from τ_{i1} to τ_{ik} , and ρ_{ik} are individual paths in Π_{ik} . There may exist multiple paths to a particular vertex; hence, the computed offset must consider the worst-case scenario over all possible paths leading to that vertex. This amounts to the summation of WCETs of tasks (C_{ij}) through the longest path (the *max* operation). For example, there are two paths that lead to the destination τ_{16} in Figure 2a. Assuming a stage delay of one time unit, the offset for τ_{16} is the maximum between path $\langle \tau_{11}, \tau_{12} \rangle$ and $\langle \tau_{11}, \tau_{13}, \tau_{14}, \tau_{15} \rangle$. Using Equation 1, we select the worst-case offset between 13 and 14, respectively, leading to an ϕ_{16} that is equal to 14.

6.2 ComputeJitters: Computing Jitters

Similar to offsets, we compute the release jitter of every task (excluding the root task) by incorporating the WCRTs

of preceding tasks on the path, and the application's release jitter. The release jitter J_{ik}^R of task τ_{ik} is defined as follows:

$$J_{ik}^R = \max_{\forall \rho_{ij} \in \Pi_{ik}} \left(\sum_{m=1}^{\text{Path Tail}} R_{im} \right) - \phi_{ik} \quad (2)$$

Equation 2 selects the maximum WCRT of the sum of WCRTs in every path from the source task to the task of interest; thereby, selecting the largest of the WCRT contributions.

7 OFLA

We introduce the theory of an offset-based FLA, which includes precedence relationships between computation and communication tasks. OFLA computes offset and jitters using holistic analysis (Section 6). Unlike prior works [13], [7], OFLA carefully considers offsets when there exists indirect interference from tasks as well. This is necessary because offsets depend on the WCRT contribution of interfering tasks.

7.1 Exact Analysis

Let a communication task under analysis τ_{ab} traverse a path δ_{ab} of communication resources. Given offsets for each communication task in the path δ_{ab} , a **critical instant** is the simultaneous activation of τ_{ab} with a communication task with equal or higher priority amongst all applications. The total number of critical instants we consider (including τ_{ab} itself) is: $N_v(\tau_{ab}) = (|S_a^D(\tau_{ab})| + 1) \cdot \prod_{\forall i \neq a} |S_i^D(\tau_{ab})|$ where $S_i^D(\tau_{ab})$ is the set of higher priority tasks in application i that directly interfere with τ_{ab} on its path δ_{ab} . Each of the $N_v(\tau_{ab})$ variations is characterized by a tuple v of indices, one for each application. Each index $v(i)$ identifies the task of application A_i that initiates the critical instant. To discover the critical instant that contributes most to the WCRT, we apply the categorization of instances of an interfering task τ_{ij} as done by the authors in [2]:

- *Set 0*: Activations that occur before the critical instant and cannot occur within the busy period even after experiencing maximum jitter.
- *Set 1*: Activations that occur at or before the critical instant and can be delayed by some jitter such that they coincide with the critical instant.
- *Set 2*: Activations that occur after the critical instant.

Note that the largest contribution to the WCRT occurs: 1) when activations of Set 1 suffer jitter such that they all occur at the critical instant, and 2) when activations in Set 2 have zero jitter. The worst-case contribution of an application A_i to the critical instant of task τ_{ab} is obtained when the first activation of a task τ_{ik} in $S_i^D(\tau_{ab})$ that occurs within the busy period coincides with the critical instant after suffering maximum jitter. Further details are in [2].

The phase is the time difference between the critical instant and the first activations in Set 2. The phase φ_{ijk} between any task τ_{ij} and the critical instant created by task τ_{ik} is given by:

$$\varphi_{ijk} = T_i - (\phi_{ik} + J_{ik}^R + J_{ik}^I - \phi_{ij}) \bmod T_i$$

where ϕ_{ij} is the reduced offset of τ_{ij} to the period 0 to T_i , and is given by: $\phi_{ij} = \Phi_{ij} \bmod T_i$. Here, J_{ik}^I is the interference jitter suffered by τ_{ik} and is given by:

$$J_{ik}^I = R_{ik} - C_{ik} - J_{ik}^R - \phi_{ik}$$

The interference jitter J_{ik}^I represents the interference suffered by τ_{ik} only due to tasks in the indirect interference set of task τ_{ab} , $S_i^I(\tau_{ab})$.

Theorem 1 specifies the worst-case interference that task τ_{ab} suffers from an application A_i when the critical instant coincides with task τ_{ik} . It considers higher priority tasks in the direct interference set of τ_{ab} , and it includes tasks in the indirect interference set as interference jitter.

Theorem 1: The worst-case contribution of an application A_i on the response time of task τ_{ab} at time t when the critical instant coincides with the activation of task τ_{ik} is given by:

$$W_{ik}(\tau_{ab}, t) = \sum_{\forall j \in S_i^D(\tau_{ab})} \left(\left\lceil \frac{J_{ij}^R + J_{ij}^I + \varphi_{ijk}}{T_i} \right\rceil + \left\lceil \frac{t - \varphi_{ijk}}{T_i} \right\rceil \right) * C_{ij}$$

For each variation v , we find the completion time of each job instance p in the busy period of task τ_{ab} . The completion time $w_{ab}^v(p)$ of job p is given by:

$$w_{ab}^v(p) = (p - p_{0,ab}^v + 1) * C_{ab} + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, w_{ab}^v(p)) \quad (3)$$

where $p_{0,ab}^v$ is the number of the lower-numbered job and is equal to $\left\lceil \frac{J_{ab}^R + \varphi_{abv(a)}}{T_a} \right\rceil + 1$. To obtain the maximum p , $P_{B,ab}^v$, we replace P in Equation 3 with $P_{B,ab}^v = \left\lceil \frac{B_{ab}^v - \varphi_{abv(a)}}{T_a} \right\rceil$, and $w_{ab}^v(p)$ with B_{ab}^v . Hence, we obtain the busy period B_{ab}^v and $P_{B,ab}^v$. Instances $p_{0,ab}^v$ to $P_{B,ab}^v$ of τ_{ab} are used to find the maximum response time. To obtain the global response time of job instance p of τ_{ab} for a particular variation v , we subtract from the completion time the activation time of job p . The activation of τ_{ab} occurs Φ_{ab} after the event arrival, and activation of job p occurs at $\varphi_{abv(a)} + (p - 1) * T_a$ measured from the critical instant. Therefore, the global WCRT is given by:

$$R_{ab}^v(p) = w_{ab}^v(p) - \varphi_{abv(a)} - (p - 1) * T_a + \Phi_{ab}$$

The WCRT of task τ_{ab} is then the maximum amongst all jobs across all variations and is given by:

$$R_{ab} = \max_{\forall v} \left(\max_{p=p_{0,ab}^v \dots P_{B,ab}^v} R_{ab}^v(p) \right)$$

7.2 Direct and Indirect Interferences

Theorem 1 presents the worst-case contribution of application A_i on the response time of τ_{ab} by including the interference jitter suffered by τ_{ij} . However, it is possible that for a particular variation of critical instants, the offsets between indirectly interfering tasks τ_{ij} and the task initiating the critical instant τ_{ik} is such that τ_{ij} does not contribute to the busy period of τ_{ab} at all. Hence, if τ_{ij} directly interferes instead of indirectly interferes, the computed WCRT would be less. In

such cases, viewing the communication network as a unique resource with all higher priority tasks directly interfering with task τ_{ab} will result in a lower WCRT than OFLA. Therefore, the execution time of τ_{ab} is bounded by the least WCRT from considering each indirectly interfering task as either direct or indirect interference.

The implication of considering each task causing indirect interference as either in the direct or indirect interference sets results in an exponential number of combinations. If the number of indirectly interfering tasks is m , the number of combinations that need to be checked is equal to 2^m for each variation v . Assuming that $R_{ab}^{v,\alpha}(p)$ is the WCRT of job p of task τ_{ab} for a given variation v and a combination α , then the overall WCRT is given by:

$$R_{ab} = \max_{\forall v} (\min_{\forall \alpha} (\max_{p=p_{0,ab}^v \dots P_{B,ab}^v} R_{ab}^{v,\alpha}(p)))$$

7.3 An Approximated Analysis

The exact analysis is exponential in the number of variations ($N_v(\tau_{ab})$) of critical instants, and the number of tasks indirectly interfering with τ_{ab} . Suppose m is the number of indirectly interfering tasks, then the computational complexity is $N_v(\tau_{ab}) * 2^m$. For practical and tractability reasons, we discover and present an approximated analysis.

7.3.1 Approximation 1 (Apx1)

We obtain an upper bound on the interference of tasks of an application A_i in a busy period of duration w as the maximum of all possible interferences that could be caused by considering each of the tasks of A_i as the one initiating the busy period [2]:

$$W_i^*(\tau_{ab}, w) = \max_{\forall k \in S_i^D(\tau_{ab})} W_{ik}(\tau_{ab}, w)$$

We still consider all possibilities of critical instants from all higher priority tasks $S_a^D(\tau_{ab})$ within the application A_a and from task τ_{ab} itself to reduce the pessimism. The completion time $w_{abc}(p)$ of job p of task τ when the critical instant coincides with the activation of task τ_{ac} is given by:

$$w_{abc}(p) = (p - p_{0,abc} + 1) * C_{ab} + W_{ac}(\tau_{ab}, w_{abc}(p)) + \sum_{\forall i \neq a} W_i^*(\tau_{ab}, w_{abc}(p)) \quad (4)$$

where $p_{0,abc}$ is the number of the lower-numbered job and is equal to $\left\lceil \frac{J_{ab}^R + \varphi_{abc}}{T_a} \right\rceil + 1$. The maximum value of p that we need to check is $P_{B,abc} = \left\lceil \frac{B_{abc} - \varphi_{abc}}{T_a} \right\rceil$. We obtain the busy period B_{abc} by replacing $w_{abc}(p)$ and p with B_{abc} and $P_{B,abc}$ in Equation 4, respectively.

The global WCRT for a particular critical instant and a combination of indirectly interfering tasks is given by:

$$R_{abc}(p) = w_{abc}(p) - \varphi_{abc} - (p - 1) * T_a + \Phi_{ab}$$

7.3.2 Approximation 2 (Apx2)

The second approximation is for the combinations of m indirect interference tasks that we need to check. Instead of checking all 2^m combinations we only check two: α_1 – all m tasks are in the indirect interference set $S^I(\tau_{ab})$, and α_2 – all m tasks are in the direct interference set $S^D(\tau_{ab})$. Notice that α_2 is equivalent to viewing all communication across a unique resource.

For each critical instant we take the minimum WCRTs of the two combinations that we check. The WCRT of task τ_{ab} is then obtained by taking the maximum across all critical instants. The approximated analysis is polynomial and considers a number of response times equal to two times the number of higher priority tasks in application A_a plus one (for including task τ_{ab} itself), i.e., $(|S_a^D(\tau_{ab})| + 1) * 2$. The overall WCRT is thus given by:

$$R_{ab} = \max_{\forall c \in S_a^D(\tau_{ab}) \cup b} (\min_{\alpha_1, \alpha_2} (\max_{p=p_{0,ab}^v \dots p_{B,ab}^v} R_{ab}^{v,\alpha}(p)))$$

8 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.

8.1 Direct and Indirect Interference Sets

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

Definition 1: Direct interference. A communication task τ_{ab} suffers direct interference from task τ_{ij} on stage s of its CR δ_{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 τ_{ab} on stage s of its CR δ_{ab} is $S_i^D(\tau_{ab}) = \{\tau_{ij} \mid \forall \tau_{ij} \in \Gamma_i^M, s \in \delta_{ab} \cap \delta_{ij} \wedge (P_{ab} < P_{ij})\}$.

Task τ_{ab} suffers indirect interference from task τ_{kl} when task τ_{ab} has direct interference with task τ_{ij} which has direct interference with task τ_{kl} ; however, tasks τ_{ab} and τ_{kl} do not directly interfere with each other. Note also that for τ_{kl} to indirectly interfere with τ_{ab} , the direct interference between τ_{ij} and τ_{kl} must occur before τ_{ij} interferes with τ_{ab} . The indirect interference set $S_{ij}^I(\tau_{ab})$ is the set of tasks indirectly interfering with τ_{ab} through task τ_{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 δ_{ij} .

Definition 3: Indirect interference. A communication task τ_{ab} suffers indirect interference from task τ_{kl} if and only

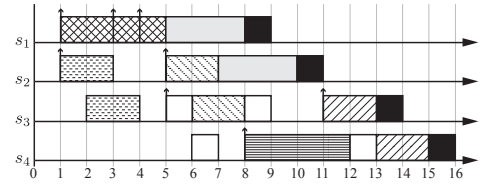


Fig. 3: An example schedule. Up arrows are release times, and the task under analysis τ_{abc} is in black.

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 τ_{ab} through task τ_{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 τ_{ab} , but they must still be considered in the analysis because they can delay τ_{ij} . We account for such interference by computing an indirect interference jitter term $J_{ij}^I(\tau_{ab})$ for τ_{ij} in Section 8.4. Similarly to [5], 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 τ_{ij} before reaching the first stage on which it causes interference on τ_{ab} .

8.2 Derivation of a Response Time Estimate

We focus on deriving an upper-bound $\hat{R}$ to the response time of the task under analysis τ_{ab} . Since indirect interferences are accounted for by indirect interference jitter, we only consider directly interfering tasks. Let $\{s_1, \dots, s_M\}$ be the ordered set of stages traversed by τ_{ab} along its path δ_{ab} where M is the number of stages in δ_{ab} . Note that if task τ_{ij} in $S_i^D(\tau_{ab})$ interferes with τ_{ab} on a non-contiguous set of stages, then τ_{ij} must be split into two or more directly interfering tasks such that each of the new tasks interferes with τ_{ab} on a contiguous set of stages.

In our derivation, we first consider the transmission of any job τ_{abc} of task τ_{ab} in any valid schedule $\Theta(t, s_k)$ on all stages of δ_{ab} . We discuss how to compute an upper-bound $\hat{R}$ to the response time of τ_{abc} for that specific schedule; for clarity, we measure the response time from the activation time of τ_{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 τ_{ab} itself. Finally, we show that independent of the schedule (e.g., $\Theta(t, s_k)$) and the specific job instance c of τ_{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 τ_{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 5, we consider that jobs of

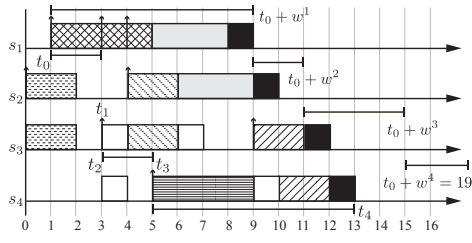


Fig. 4: An example of the stage-normalized schedule.

the same communication task are transmitted in FIFO order. Hence, when analyzing the job under analysis τ_{abc} , we simply assume that the priority of any job $\tau_{abc'}$ that follows τ_{abc} , i.e. with $c' > c$, is lower than the priority of τ_{abc} . The second transformation involves the schedule $\Theta(t, s_k)$. Note that if any job τ_{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 δ_{ab} , the corresponding stage-normalized schedule is $\tilde{\Theta}(t, s_k) = \Theta(t + k - 1, s_k)$.

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 2 and 4, 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 $\tilde{\Theta}$ if the stage-normalized schedule continuously transmits jobs with priority greater than or equal to $\mathbb{P}_{abc}$ in slots $\langle t, \dots, t' - 1 \rangle$ and τ_{abc} is not completed before slot $t' - 1$. 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 τ_{abc} in every stage s_k of δ_{ab} . For simplicity and since lower-priority jobs do not affect the schedule of τ_{abc} in any way, assume that all jobs in Figure 4 have higher priority than the job under analysis. Similarly, note that by definition, jobs that arrive on stage s_k after τ_{abc} finishes executing on that stage do not need to be included in the busy interval. 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 τ_{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 τ_{abc} only once.

Definition 7: Busy chain. Let t_M be the time at which the job under analysis τ_{abc} finishes executing on its last stage s_M in $\tilde{\Theta}$, i.e., its last datum is transmitted in slot $t_M - 1$ (interval $[t_M - 1, t_M)$). For each stage s_k in δ_{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, $\langle [t_0, t_1), \dots, [t_{M-1}, t_M) \rangle$ is the busy chain for τ_{abc} in $\tilde{\Theta}$.

It is easy to see that every job τ_{abc} admits a unique busy chain for a given stage-normalized schedule $\tilde{\Theta}$. 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 $\tilde{\Theta}$ 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 $t_0, \dots, t_3$ in stages $s_1, \dots, s_4$, 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 $t_1 = t_2 = 3$. Finally, to show that the definition is consistent, we prove that the busy chain always contains the execution of τ_{abc} , i.e., the finishing time of τ_{abc} on any stage s_k is no earlier than the start time t_{k-1} of the busy chain on that stage:

Lemma 1: Let t_k^f be the time at which the job under analysis τ_{abc} finishes executing on stage s_k in $\tilde{\Theta}$. Then $t_k^f \geq t_{k-1}$.

Proof: The lemma proceeds by induction on the stage number k , starting from $k = M$ and proceeding backwards until $k = 1$. Note that by Definition 7, $t_k^f = t_M \geq t_{M-1}$, hence the base case immediately follows.

Assuming that the hypothesis holds for each stage down to s_{k+1} , we now show that it also holds for stage s_k . By definition of a busy chain, the schedule on s_{k+1} is $\mathbb{P}_{abc}$ -level idle in $[t_k - 1, t_k)$; since $t_{k+1}^f \geq t_k$, this implies that no job with priority greater than or equal to τ_{abc} is transmitted in slot $t_k - 1$ on s_{k+1} . Hence, all data of τ_{abc} must be transmitted on s_k in slots t_k or after; otherwise, a datum of τ_{abc} would be transmitted in slot $t_k - 1$ on stage s_{k+1} and the slot would not be $\mathbb{P}_{abc}$ -level idle. This implies $t_k^f \geq t_k \geq t_{k-1}$, concluding the proof. $\square$

Let $\hat{t}$ be the activation time of τ_{abc} . The response time of τ_{abc} in $\tilde{\Theta}$ 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 τ_{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 2: Consider the busy chain of τ_{abc} in $\bar{\Theta}$. A given datum of any job τ_{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: The proof is similar to the induction step for Lemma 1. Note that since τ_{ijp} is executed in the busy chain, it must hold that $\mathbb{P}_{ijp} > \mathbb{P}_{abc}$. Consider a datum of τ_{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 $[t_l - 1, t_l)$; since $t'_{l+1} \geq t_l$ by Lemma 1, this implies that no job with priority greater than or equal to τ_{abc} is transmitted in slot $t_l - 1$ on s_{l+1} . 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. $\square$

Intuitively, Lemma 2 implies that every datum of an interfering job τ_{ijp} needs to be counted only once towards the length of the busy chain of τ_{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 3 and 4. The following two lemmas provide a way to compute the length of interfering jobs on each stage.

Definition 8: Interfering job set. S_k^J is the set of all jobs with priority higher than or equal to $\mathbb{P}_{abc}$ that are transmitted on stage s_k , with the exclusion of τ_{abc} itself.

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

Note that S_k^J does not include τ_{abc} , but it includes any previous jobs of τ_{ab} .

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 3 (base case). We then compute an upper bound to $[t_0, t_k)$ for each stage s_k in Lemma 4 (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 S_k^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 $\bar{W}_1^{S^J}, \dots, \bar{W}_k^{S^J}$ to compute such sum (note that workloads are defined based on release times of jobs, not when they are transmitted); furthermore, since Lemma 2

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 3: Consider the busy chain of τ_{abc} in schedule $\bar{\Theta}$. Then for any value Δ such that:

$$\Delta = L_{ab} + \bar{W}_1^{S^J}(t_0, t_0 + \Delta), \quad (5)$$

Δ is an upper-bound to $t_1 - t_0$.

Proof: Note that by definition of a busy chain, the schedule is $\mathbb{P}_{abc}$ -level busy on s_1 in $[t_0, t_1)$ and $\mathbb{P}_{abc}$ -level idle in $[t_0 - 1, t_0)$. Hence, no job in S_1^J released on s_1 before t_0 can be transmitted in slot t_0 or after. Furthermore, clearly no job in S_1^J released at or after $t_0 + \Delta$ can be transmitted in $[t_0, t_0 + \Delta)$. Therefore, $\Delta = L_{ab} + \bar{W}_1^{S^J}(t_0, t_0 + \Delta)$ is the sum of the transmission time of all jobs in S_1^J that can be executed in $[t_0, t_0 + \Delta)$, plus τ_{abc} itself. We consider three possible cases: 1) there is a $\mathbb{P}_{abc}$ -level idle slot in $[t_0, t_0 + \Delta)$. Then by definition $t_1 < t_0 + \Delta$. 2) The schedule is $\mathbb{P}_{abc}$ -level busy in $[t_0, t_0 + \Delta)$ and τ_{abc} is released in $[t_0, t_0 + \Delta)$: then it follows that τ_{abc} must finish exactly at $t_0 + \Delta$ and the schedule is $\mathbb{P}_{abc}$ -level idle after $t_0 + \Delta$, implying $t_1 = t_0 + \Delta$. 2) The schedule is $\mathbb{P}_{abc}$ -level busy in $[t_0, t_0 + \Delta)$ and τ_{abc} is not released in $[t_0, t_0 + \Delta)$: this is impossible, since $\bar{W}_1^{S^J}(t_0, t_0 + \Delta) < \Delta$ implies that there are not enough data units of jobs in S_1^J to transmit continuously in the interval $[t_0, t_0 + \Delta)$ without considering τ_{abc} itself. In summary, considering Cases 1) and 2) we have $t_1 \leq t_0 + \Delta$, concluding the lemma. $\square$

Since we are interested in the tightest possible upper-bound to the response time of τ_{abc} , we simply compute the minimal value w^1 of Δ for which Lemma 3 holds as:

$$w^1 = \min\{\Delta | \Delta = \bar{W}_1^{S^J}(t_0, t_0 + \Delta)\}. \quad (6)$$

As an example, when we apply Lemma 3 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 . It is easy to see that the computed bound must have finite length as long as the sum of the utilization (e.g., L_{ij}/T_i) of tasks of jobs in S_k^J is less than one; therefore, under such assumption we can always compute a valid value for w^1 .

Lemma 4: For $1 \leq k \leq M$, w^k is an upper-bound to $t_k - t_0$, where:

$$w^k = \min\{\Delta | \Delta = \bar{W}_k^{S^J}(t_0, t_0 + \Delta) + w^{k-1} - \bar{W}_{(S_{k-1}^J \cap S_k^J)}(t_0, t_0 + w^{k-1})\} \quad (7)$$

Proof: Note that for $k = 1$, Equation 7 reduces to Equation 6. Also, the intersection $(S_{k-1}^J \cap S_k^J)$ 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 S_{k-1}^J released in $[t_0, t_0 + w^{k-1})$ plus τ_{abc} . By Lemma 3 and definition of

$\bar{W}_{S^J}(t, t')$, this is true for $k - 1 = 1$ (base case). We need to prove that it holds for stage s_k (induction step).

We show that the maximum sum of transmission lengths of τ_{abc} plus jobs transmitted in the busy chain on $s_1, \dots, s_k$ in $[t_0, t_0 + w^k + 1)$ is w^k ; we can then use the same three cases as in the proof of Lemma 5 to prove that w^k is a valid upper bound to $t_k - t_0$. No job in S_k^J released at or after $t_0 + w^k$ can be transmitted in $[t_0, t_0 + w^k)$. We will next prove that no job in S_k^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 2, any datum of a job contributing to the busy chain on stages $s_1, \dots, s_{k-1}$ can not contribute to the busy chain on s_k . Therefore, we can upper-bound the sum of the transmission lengths on τ_{abc} plus jobs transmitted in the busy chain in $[t_0, t_0 + w^k + 1)$ by taking the workload $\bar{W}_{S_k^J}(t_0, t_0 + w^k)$, summing the maximum length of the busy chain w^{k-1} up to stage s_{k-1} (which includes τ_{abc}), and subtracting the transmission time of jobs that are released in S_k^J within $[t_0, t_0 + w^k + 1)$ but were already counted in w^{k-1} , which is $\bar{W}_{(S_{k-1}^J \cap S_k^J)}(t_0, t_0 + w^{k-1})$; this is equivalent to computing w^k according to Equation 7. This concludes the induction step, sincere have also shown that w^k indeed includes the transmission time of all jobs in S_k^J released in $[t_0, t_0 + w^k)$ plus τ_{abc} .

We still need to prove that no job in S_k^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 $\tau_{ijp} \in S_k^J$ 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 τ_{ijp} is transmitted on s_l . Then it must be $t'_l < t_{l-1} - 1$, otherwise, τ_{ijp} would be transmitting during the $\mathbb{P}_{abc}$ -idle slot $[t_{l-1} - 1, t_{l-1})$ given that $t'_l \geq t_{l-1}$ according to Lemma 1. We then use the same reasoning as in Lemma 2 to show that $t'_k < t_{k-1} - 1$, where t'_k is the slot during which the last datum of τ_{ijp} is transmitted on s_k . $\square$

Figure 4 shows the values of $w^1, \dots, w^4$ computed for the figure's schedule. Let us consider $w^2 = \bar{W}_{S_2^J}(t_0, t_0 + w^2) + w^1 - \bar{W}_{(S_1^J \cap S_2^J)}(t_0, t_0 + w^1)$. As previously discussed, w^1 includes the transmission times of all jobs on s_1 including the task under analysis τ_{abc} (in solid black). $\bar{W}_{S_2^J}(t_0, t_0 + w^2)$ 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 $\bar{W}_{(S_1^J \cap S_2^J)}(t_0, t_0 + w^1)$ 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 $w^1, \dots, w^4$ 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 τ_{ab} .

Comparison with Previous Work

Note that compared to our previous work [?], our new ORTAP analysis employs a different definition of $\mathbb{P}_{abc}$ -level busy interval. More in details, the analysis in [?] included execution by jobs with priority greater than or equal to τ_{abc} that were transmitted after τ_{abc} 's completion; as a result, the analysis in [?] is more pessimistic. As an example, assume that a job with higher priority than τ_{abc} and executing on stage s_1 only is activated at time $t = 9$ in the schedule $\bar{\Theta}$ in Figure 4. Since the schedule on stage s_1 is continuously busy until the completion of such higher priority job, in our original analysis it would be included in the busy interval, even if it clearly cannot delay τ_{abc} . As a result, the latency bounds computed in [?] can be worse than the bounds obtained by OFLA. On the other hand, in Section ... we prove that our new ORTAP analysis strictly dominates OFLA.

8.3 Critical Activation Patterns

Lemma 4 gives us a way to compute an upper-bound $\bar{R}$ on the response time of τ_{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 τ_{abc} in the stage-normalized schedule $\bar{\Theta}$. We can compute the response time $\hat{R}$ in the original schedule Θ as $\hat{R} = \bar{R} + M - 1$. Finally, since $\hat{R}$ and $\bar{R}$ are measured from the activation time of τ_{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 τ_{abc} of τ_{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 3 and 4.

Lemma 5: Consider applying the following rules to the release pattern of jobs in $\bar{\Theta}$:

- 1) Every job τ_{ijp} that is activated before t_0 and can be released at or after t_0 is released at t_0 .
- 2) Every job τ_{ijp} that is activated after t_0 is released immediately at its activation time.
- 3) The activation time of every application τ_i is moved earlier in time until one job of τ_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 3 and 4. 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 $\bar{W}_{S_k^J}(t_0, t_0 + \Delta)$ for any set S_k^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 τ_{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 τ_{abc} to occur earlier but not later in time. $\square$

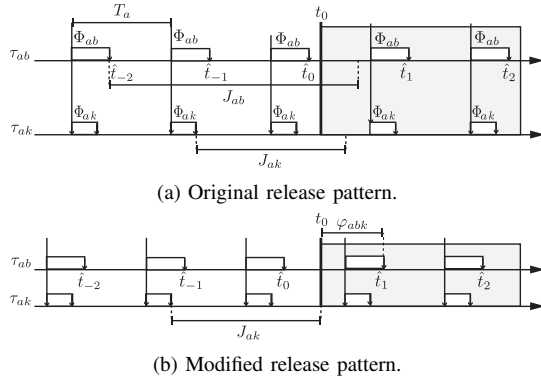


Fig. 5: Release patterns of jobs of application A_a .

Figure 5a shows an example timeline including t_0 , the beginning of a busy chain. It also shows activation times (down arrows) $\hat{t}_{-2}, \dots, \hat{t}_2$ for several jobs of τ_{ab} , as well as jobs of another higher priority task τ_{ak} of application A_a . Note that the first job of τ_{ak} cannot be released at or after t_0 . Rule 1 applies to the first three jobs of τ_{ab} and the second and third job of τ_{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 τ_{ak} is released at t_0 after suffering maximum jitter.

We call the modified pattern obtained in Lemma 5 a critical pattern. Let $\tau_{abc'}$, with $c' \leq c$, be the first job of τ_{ab} released at or after t_0 . The number of critical patterns for τ_{abc} where $\tau_{abc'}$ is the first such job is then $(|S_a^D(\tau_{ab})| + 1) \cdot \prod_{i \neq a} |S_i^D(\tau_{ab})|$. 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 $|S_a^D(\tau_{ab})| + 1$ 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 $|S_i^D(\tau_{ab})|$ 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 5 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 τ_{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 τ_{abc} of τ_{ab} as long as we vary the number of jobs of τ_{abc} that are released in $[t_0, t_0 + w^M]$.

8.4 Indirect Interference Jitter

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

$$\varphi_{ijk} = T_i - (\phi_{ik} + J_{ik}^R + J_{ik}^I(\tau_{ab}) - \phi_{ij}) \bmod T_i$$

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

$$J_{ik}^I(\tau_{ab}) = R_{ik}(\tau_{ab}) - L_{ik} - J_{ik}^R - \phi_{ik}$$

The interference jitter $J_{ik}^I(\tau_{ab})$ represents the interference suffered by flow τ_{ik} only due to flows in the indirect interference set of task τ_{ab} , $S_{ik}^I(\tau_{ab})$. We use the notation $R_{ik}(\tau_{ab})$ to denote the response time of task τ_{ik} only due to interference from higher priority tasks in the set $S_{ik}^I(\tau_{ab})$. As an example, Figure 5b shows the phase $\varphi_{abv(a)} = \varphi_{abk}$ between τ_{ab} and the beginning of the busy chain created by τ_{ak} , assuming that $J_{ak} = J_{ak}^R + J_{ak}^I(\tau_{ab})$. Note that for the first job of τ_{ab} activated after t_0 , φ_{abk} is exactly equal to the length of the interval $\hat{t} - t_0$ used in Section 8.3.

8.5 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 $p_{0,ab}^v = -\lfloor \frac{J_{ab}^R + \varphi_{abv(a)}}{T_a} \rfloor + 1$.

Lemma 6: The worst-case contribution of an application A_i to the busy chain for τ_{ab} on stage s_l when the activation of task τ_{ik} coincides with the critical instant is given by:

$$W_{ik}(\tau_{ab}, l, t) = \sum_{\forall j \in S_i^D(\tau_{ab})} \left(\left\lfloor \frac{J_{ij}^R + J_{ij}^I(\tau_{ab}) + \varphi_{ijk}}{T_i} \right\rfloor + \left\lfloor \frac{t - \varphi_{ijk}}{T_i} \right\rfloor + 1 \right) * L_{ij}$$

Proof: By definition, a workload $\bar{W}_{S^J}^J(t, t')$ includes the transmission times of all jobs in the set S_l^J that are released in the interval $[t, t']$. The set S_l^J includes all jobs of higher priority tasks of application A_i , $S_i^D(\tau_{ab})$, that contribute to the workload on stage s_l . Consider an arbitrary higher priority task of application A_i , τ_{ij} . According to Rule 1 of Lemma 5, all job instances of τ_{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 5. The job instances belonging to Rule 2 are a series of periodic activations starting φ_{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. $\square$

We further explain Lemma 6 using Figure 5b applied to task τ_{ak} , where $v(a) = k$. The first term $\lfloor \frac{J_{ak}^R + J_{ak}^I(\tau_{ab}) + \varphi_{akk}}{T_a} \rfloor$ 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 τ_{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 $S_{l-1}^D(\tau_{ab}) \cap S_l^D(\tau_{ab})$. We also use $p_{B,ab}^v$ to denote the largest-numbered job instance in the interval w^l .

Lemma 7: For each activation pattern v , the worst-case length of the busy chain for each job p of task τ_{ab} up to stage s_l is determined by:

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

where $p' = p$ if $p \leq p_{l-1,ab}^v$, else $p' = p_{l-1,ab}^v$.

Proof: This proof directly descends from Lemma 4. In the above equation, if we remove the restriction to a specific job instance p , then we compute $w_{l,ab}^v$ which is the length of the busy chain up to stage s_l where $c = p_{B,ab}^v$. Thus, computing w^l as introduced in Lemma 4. The only difference is that $w_{l,ab}^v$ separates interference from higher priority tasks and jobs from the same task into different terms. In order to compute the $w_{l,ab}^v(p)$ 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 4, and using Lemma 6, to compute $w_{l,ab}^v(p)$ we use $w_{l-1,ab}^v(p')$ 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 τ_{ab} only up till p . Note that we use p' instead of p in $w_{l-1,ab}^v(p')$. This is due to the fact that a certain p might only exist on stage s_l but not s_{l-1} , making $w_{l-1,ab}^v(p)$ 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., $p_{l-1,ab}^v$. $\square$

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

To obtain $p_{B,ab}^v$ for any stage s_l , we simply replace $w_{l,ab}^v(p)$ and $w_{l-1,ab}^v(p')$ by $w_{l,ab}^v$ and $w_{l-1,ab}^v$, i.e., we compute a busy interval instead of a response time of a specific job instance.

We also replace p by $(\lfloor \frac{w_{l,ab}^v - \varphi_{abv(a)}}{T_a} \rfloor - \lfloor \frac{w_{l-1,ab}^v - \varphi_{abv(a)}}{T_a} \rfloor)$. This simply accounts for new job instances of τ_{ab} that exist on stage s_l but not s_{l-1} .

Theorem 2: The worst-case response time of task τ_{ab} is obtained by:

$$R_{ab} = \max_{\forall v} \left(\max_{p=p_{0,ab}^v \dots p_{B,ab}^v} (R_{M,ab}^v(p)) \right) + M - 1$$

where $R_{M,ab}^v(p) = w_{M,ab}^v(p) - \varphi_{abv(a)} - (p - 1) * T_a + \Phi_{ab}$.

Proof: Using Lemma 7, we can find the worst-case length of the busy chain for each job p on stages along the path δ_{ab} of τ_{ab} for a particular activation pattern v . We also showed that the response time for a job of τ_{ab} is $\bar{R} = w^M - (\hat{t} - t_0)$, i.e., $\bar{R} = w_{M,ab}^v(p) - \varphi_{abv(a)} - (p - 1) * T_a$ for job p . To obtain the response time $R_{M,ab}^v(p)$ measured from the activation of application A_a , rather than the activation of τ_{ab} , we also need to add the offset Φ_{ab} . Next, we compute the maximum worst-case response time across all job instances, which gives us the worst-case response time of τ_{ab} in v . From Lemma 5, we consider the maximum worst-case response time across all activation patterns to obtain the WCRT of task τ_{ab} , R_{ab} . $\square$

8.6 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 $W_i^*(\tau_{ab}, l, t) = \max_{\forall k \in S_i^D(\tau_{ab})} W_{ik}(\tau_{ab}, l, t)$ 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 τ_{ab} belongs. The number of critical activation patterns that we must consider is thus reduced to $|S_i^D(\tau_{ab})| + 1$. In what follows, we derive a polynomial WCRTA for our pipelined communication model. For clarity of presentation, we use $W_i^{**}(\tau_{ab}, l, t) = \max_{\forall k \in S_i^D(\tau_{ab})} W_{ik}''(\tau_{ab}, l, t)$ where $W_{ik}''(\tau_{ab}, l, t)$ is the worst-case contribution of an application A_i to the busy chain for task τ_{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 $S_{l-1}^D(\tau_{ab}) \setminus S_l^D(\tau_{ab})$.

Lemma 8: For a critical instant created with task τ_{ac} , the worst-case length of the busy chain for each job p of task τ_{ab}

up to stage s_l is determined by:

$$\begin{aligned} w_{abc}(p) = & \sum_{s=2 \dots l} \sum_{\forall i \neq a} W_i''(\tau_{ab}, s, w_{s-1}^{abc}(p')) \\ & + \sum_{\forall i} W_i^*(\tau_{ab}, l, w_{abc}(p)) + \sum_{s=2 \dots l} W_{ac}''(\tau_{ab}, s, w_{s-1}^{abc}(p)) \\ & + W_{ac}(\tau_{ab}, l, w_{abc}(p)) + (p - p_{0,ab}^v + 1) * L_{ab} \end{aligned}$$

Proof: We first transform the interference from Lemma 7 into a more convenient form for the discussion. Consider using Lemma 7 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 $w_{l-1}^{ab} + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, w_{l-1}^{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, w_{l-1}^{ab})$ of Lemma 7. The term w_{l-1}^{ab} can be expanded into $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, l-1, w_{l-1}^{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l-1, w_{l-2}^{ab})$. We expand all terms until reaching the first stage s_1 . Hence, we get:

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

Recall that $\sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, w_{l-1}^{ab})$ is the interference on stage s_{l-1} from tasks in the set $S_{l-1}^D(\tau_{ab}) \cap S_l^D(\tau_{ab})$. Hence the terms, $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, l-1, w_{l-1}^{ab}) - \sum_{\forall i} W'_{iv(i)}(\tau_{ab}, l, w_{l-1}^{ab})$ can be rewritten as $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, l, w_{l-1}^{ab})$ which is the interference due to tasks in the set $S_{l-1}^D(\tau_{ab}) \setminus S_l^D(\tau_{ab})$. This is intuitive since the first term is interference from the set $S_{l-1}^D(\tau_{ab})$ and the second term is the interference from the set $S_{l-1}^D(\tau_{ab}) \cap S_l^D(\tau_{ab})$. So their difference yields the interference in the set $S_{l-1}^D(\tau_{ab}) \setminus S_l^D(\tau_{ab})$. Therefore, we can write the interference in Lemma 7 in the form: $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, 2, w_{1}^{ab}) + \dots + \sum_{\forall i} W''_{iv(i)}(\tau_{ab}, l, w_{l-1}^{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, w_{l}^{ab})$. Or in a more compact form $\sum_{s=2 \dots l} \sum_{\forall i} W''_{iv(i)}(\tau_{ab}, s, w_{s-1}^{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, l, w_{l}^{ab})$.

Next, we consider the first stage on path δ_{ab} of τ_{ab} . The interference on stage s_1 is equal to $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, 1, w_{1}^{ab})$. Using the approximation introduced earlier, the interference will be equal to $\sum_{\forall i} W_i^*(\tau_{ab}, 1, w_{1}^{ab})$. 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., $W_i^*(\tau_{ab}, 1, w_{1}^{ab})$ is an upper-bound of $\sum_{\forall i} W_{iv(i)}(\tau_{ab}, 1, w_{1}^{ab})$. This is similar to the derivation in [14].

Let us consider the second stage s_2 . The interference on this stage is equal to $\sum_{\forall i} W''_{iv(i)}(\tau_{ab}, 2, w_{1}^{ab}) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, 2, w_{2}^{ab})$. 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 $\sum_{\forall i} W_i''(\tau_{ab}, 2, w_{1}^{ab}) + \sum_{\forall i} W_i^*(\tau_{ab}, 2, w_{2}^{ab})$. Since $W_i^*(\tau_{ab}, 2, w_{2}^{ab}) \geq W_{iv(i)}(\tau_{ab}, 2, w_{2}^{ab})$ and the same holds for the interference that occurs only on stage 1, $W_i''(\tau_{ab}, 2, w_{1}^{ab}) \geq W''_{iv(i)}(\tau_{ab}, 2, w_{1}^{ab})$. 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 τ_{ab} , we need to consider all possible critical instants from application A_a . $\square$

Theorem 3: The worst-case response time of task τ_{ab} is obtained by:

$$R_{ab} = \max_{\forall c \in S_i^D(\tau_{ab}) \cup b} \left(\max_{p=p_{0,ab}^v \dots p_{M^{B,ab}}^v} (R_{abc}(p)) \right) + M - 1$$

where $R_{abc}(p) = w_{abc}(p) - \varphi_{ab(a)} - (p-1) * T_a + \Phi_{ab}$.

Proof: The proof is similar to the proof of Theorem 2. $\square$

9 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 [5], [6], [7]. This NoC supports wormhole switching with flit-level preemption. Details of the NoC architecture are available in [5]. In particular, we propose two instances of NoCs with sizes 4×4 and 8×8 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) [12], and Palencia and Gonzalez's [2] (PAL) analyses, 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

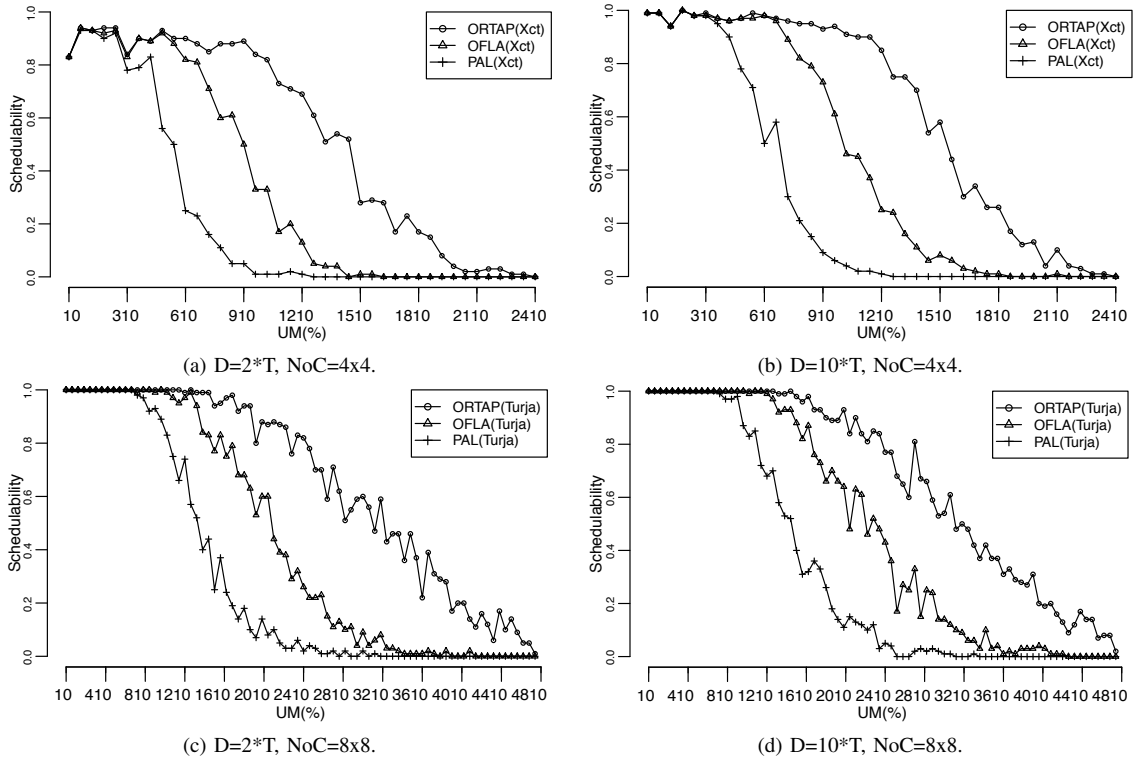


Fig. 7: Schedulability of application sets with 10 tasks per application.

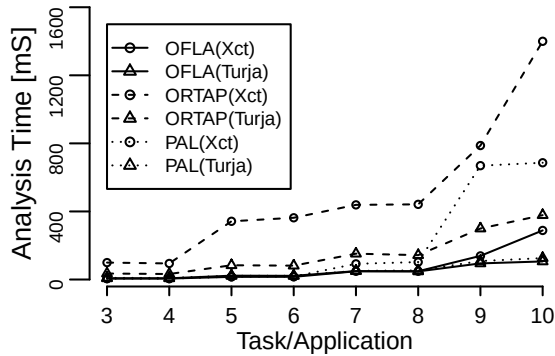


Fig. 6: Run-time comparison of various WCRTA techniques.

and jitters are calculated based on the methods presented in Section 5.3. 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 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.

10 CONCLUSION

In this work, we present a holistic analysis for computing offsets and jitters of tasks in a DAG task graph. We present

OFLA and ORTAP for computing the WCRT of applications in 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 these WCRTA techniques, 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 OFLA and ORTAP against prior work. Our results show that OFLA improves schedulability by 38.1% compared to PAL for the two NoC sizes. The results also 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] H. Kashif, S. Gholamian, and H. Patel, "SLA: A Stage-level Latency Analysis for Real-time Communication in a Pipelined Resource Model," *IEEE Transactions on Computers*, vol. PP, April 2014.
- [5] 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.
- [6] —, "Schedulability analysis and task mapping for real-time on-chip communication," *Real-Time Systems*, vol. 46, pp. 360–385, 2010.
- [7] —, "Priority assignment for real-time wormhole communication in on-chip networks," in *Real-Time Systems Symposium*, 2008, pp. 421 – 430.
- [8] 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.
- [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, "Real 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] 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 (ASP-DAC)*, SunTec, Singapore, January 2014.
- [12] 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.
- [13] Z. Shi and A. Burns, "Schedulability Analysis and Task Mapping for Real-Time on-Chip Communication," *Real-Time Syst.*, vol. 46, pp. 360–385, 2010.
- [14] K. Tindell, "Adding time-offsets to schedulability analysis," Dept. of Computer Science, University of York, Tech. Rep., January 1994.