Technical supplementary material for RefinedProsa

KIMAYA BEDARKAR, MPI-SWS, Germany
LAILA ELBEHEIRY, MPI-SWS, Germany
MICHAEL SAMMLER, ETH Zürich and ISTA, Austria
LENNARD GÄHER, MPI-SWS, Germany
BJÖRN BRANDENBURG, MPI-SWS, Germany
DEREK DREYER, MPI-SWS, Germany
DEEPAK GARG, MPI-SWS, Germany

1 FORMAL DETAILS OF THE RTA

In this section, we provide the missing details on our RTA which did not fit into the paper. We start by first expanding on the aRSA primer presented in Section 4 of the paper. We then show the response-time bounds provided by aRSA as well as our definition of the *SBF* bound. Table 1 provides a summary of the notation used in the following.

1.1 Applying the Restricted-Supply Analysis

As mentioned in the paper, we use aRSA to state an overhead-aware RTA for our system. In this section, we introduce additional details from the aRSA framework and how it applies to Rössl, and present the response-time bounds *assuming a given supply bound function* SBF_i for each task $\tau_i \in \tau$. We define the actual SBF_i characterizing all possible overheads in Rössl affecting a given task under analysis $\tau_i \in \tau$ afterwards in §1.2.

Further, recall from the paper that our RTA applies to the *release sequence* (as opposed to the arrival sequence) and defines the response-time bound with respect to a job's *release time* $r_{i,j}$ (as opposed to the job's arrival time $a_{i,j}$). Please refer to the main paper for how to obtain the bounds with respect to the original arrival sequence and arrival times. In the remainder of this text, the response times are always meant to be interpreted with respect to a job's release time.

aRSA is an analysis framework newly introduced to the Prosa library by Sergey Bozhko that generalizes the aRTA framework introduced by Bozhko and Brandenburg [1], which relies on the *busy-window principle*. Before going ahead, we briefly describe the busy-window principle since aRSA relies on it too. In the following, let τ_i denote the task under analysis.

The busy-window principle. The busy-window principle is a well-known technique in real-time systems for analyzing response-times. At a high level, the busy window for any job $\tau_{i,j}$ is an interval in which the system continuously has pending workload of at least P_i priority. Analyzing the response-time of a job inside its busy window greatly simplifies the problem of finding a reasonably accurate response-time bound. While aRSA (and aRTA) both present a very abstract and generic definition of busy windows that can be applied to a number of different systems, we focus here on an intuitive explanation of busy windows that is specific to Rössl.

In the case of Rössl, the busy window for any job $\tau_{i,j}$ is an interval $[t_1, t_2)$ such that (1) all higher-or-equal priority jobs (with respect to $\tau_{i,j}$) that have been released before t_1 have finished by t_1, (2) at every point in the interval, there is at least one higher-or-equal priority job that is

Authors' addresses: Kimaya Bedarkar, MPI-SWS, Saarland Informatics Campus, Germany, kbedarka@mpi-sws.org; Laila Elbeheiry, MPI-SWS, Saarland Informatics Campus, Germany, lelbehei@mpi-sws.org; Michael Sammler, ETH Zürich and ISTA, Klosterneuburg, Austria, michael.sammler@ist.ac.at; Lennard Gäher, MPI-SWS, Saarland Informatics Campus, Germany, gaeher@mpi-sws.org; Björn Brandenburg, MPI-SWS, Saarland Informatics Campus, Germany, bbb@mpi-sws.org; Derek Dreyer, MPI-SWS, Saarland Informatics Campus, Germany, dreyer@mpi-sws.org; Deepak Garg, MPI-SWS, Saarland Informatics Campus, Germany, dg@mpi-sws.org.

Table 1. Table of Notation

Notation	Full name	Meaning
ε	-	$\varepsilon \triangleq 1$ is the smallest unit of time
τ_i	-	The ith task
τ	-	The set of tasks in the workload
$\tau_{i,j}$	-	The jth job of the ith task
$a_{i,j}$	-	The arrival time of $\tau_{i,j}$
$r_{i,j}$	-	The release time of $\tau_{i,j}$
P_i	-	The priority of the ith task
α_i	Arrival curve	The bound on the maximum rate of arrivals for any task τ_i
β_i	Release curve	The derived bound on the maximum rate of releases for any task τ_i
RBF_i	Request Bound Function	Bound on the supply demanded by a task τ_i in a given interval
SBF_i	Supply Bound Function	A lower-bound on the supply guaranteed to be provided by the system in the busy window of τ_i
$\mathcal{W}_i$	-	Bound on the workload in the busy window of $\tau_{i,j}$
L_i	-	Bound on the length of the busy window for any job $\tau_{i,j}$
PB	Polling Bound	WCET for *PollingOvh* $\tau_{i,j}$ for any $\tau_{i,j}$
RB	Reading Bound	WCET for *ReadOvh* $\tau_{i,j}$ for any $\tau_{i,j}$
SB	Selection Bound	WCET for *SelectionOvh* $\tau_{i,j}$ for any $\tau_{i,j}$
DB	Dispatch Bound	WCET for *DispatchOvh* $\tau_{i,j}$ for any $\tau_{i,j}$
CB	Completion Bound	WCET for *CompletionOvh* $\tau_{i,j}$ for any $\tau_{i,j}$
IB	Idling Bound	Bound on how long the system is allowed to idle after new jobs have entered the system
C_i	Task Cost	WCET for *Executes* $\tau_{i,j}$ for any τ_i
TRB	Total Reading Bound	Bound on the total time spent in reading states for any interval
NRB	Non-reading bound	Bound on the total time spent in any blackout states (except for *ReadOvh*) in any interval
R_Off	Reading offset	Bound on time between a job being read and the job arriving
RPB	Reading period bound	Bound on the maximum time spent by the system in doing reads continuously

pending, (3) the job $\tau_{i,j}$ is released within $[t_1, t_2)$ (*i.e.*, $t_1 \leq r_{i,j} < t_2$), (4) all higher-or-equal priority jobs that are released before t_2 have finished by t_2. Given these conditions, we can infer two facts about the busy window that aid us in finding the response times. First, we do not have to consider any workload that is released before t_1 since we know that all the higher-or-equal priority workload released before t_1 has already finished. Second, by the end of the busy window, the job under consideration must have been released *and* finished. In fact, given the definition of the busy window, a bound on the length of the busy window already serves as a bound on the response time of the task under consideration, albeit a coarse one. Using these facts, we can simplify the task of finding the response-time bounds.

Note that we can use the advantages of a busy window only if we know that for the task under consideration a busy window exists. To guarantee this, aRSA defines a sufficient condition on the release curves, which we review next in a form specific to Rössl. If this condition holds, then we can guarantee that the busy window exists and has a finite length.

Defining the busy-window bound. To define the condition that guarantees the existence of busy windows of finite length, we consider the lower bound on the supply offered by the system (given by SBF_i) and the maximum supply that can be required by the workload (computed using each task's WCET and arrival curve), and state an inequality between these two quantities.

First, let us define a bound on the maximum supply that can be requested by all the tasks competing with the task under analysis in any busy window of length L. In order to do so, we first recall the definition of the request bound function RBF_i that bounds the service requested by any task τ_i in an interval Δ as

$$RBF_i(\Delta) \triangleq \beta_i(\Delta) \times C_i. \tag{1}$$

Recall that the analysis presented here is with respect to the release times (as opposed to the arrival times). Consequently, the busy windows are also defined with respect to the release sequence. Now, since the release sequence is known to be bounded by β_i, the equations presented here are defined with respect to β_i (as opposed to α_i).

Using the RBF, we define a bound on the *maximum workload* $\mathcal{W}_i(L)$ in a busy window of length L:

$$\mathcal{W}_i(L) \triangleq \max_0 \left(\{C_l - \epsilon \mid \tau_l \in \tau \land P_l < P_i\}\right) + \sum_{\tau_h \in \tau \land P_i \leq P_h} RBF_h(L),$$

where $\max_0(S) \triangleq \max(S)$ if S is non-empty or else $\max_0(S) \triangleq 0$.

Recall that, in a busy window, we have to consider the releases of only higher-or-equal priority jobs since the definition of the busy window guarantees that at every point in the busy window a higher or equal priority job is pending (which prevents the scheduling policy from dispatching a lower-priority job). However, in the beginning of the interval, there might execute a lower-priority job that started execution before the busy window began and that cannot be preempted. Therefore, in the equation above, the first summand bounds the supply consumed by any lower-priority job that started execution just before the interval started. The second summand bounds the total amount of work that needs to be done to complete all jobs of higher-or-equal priority tasks released during the interval of length L.

Using the bound $\mathcal{W}_i$, aRSA then defines the following bound on the maximum length of the busy window of any job of task τ_i.

$$L_i \triangleq \inf\{0 < L \mid \mathcal{W}_i(L) \leq SBF_i(L)\} \tag{2}$$

If L_i is finite, that is, if a finite solution L_i to the inequality $\mathcal{W}_i(L) \leq SBF_i(L)$ exists, then the least such L_i is a bound on the length of any busy window of task τ_i.

Intuitively, the right-hand side of the inequality $\mathcal{W}_i(L) \leq SBF_i(L)$ uses the supply bound function to determine how much service (*i.e.,* productive work) the system can supply *at least* in any interval of length L. The left-hand side gives a bound on the total work demanded *at most* by jobs executing in a busy window of length L. When the right-hand side surpasses the left-hand side, assuming a work-conserving scheduler (*i.e.,* a scheduler that does not idle the processor if there are pending jobs), we can conclude that the busy window has ended since there must exist a time instant such that all higher-or-equal priority jobs in the system have finished execution.

The response-time bound. If L_i is finite, then it is a bound on the length of the busy window for any job of task τ_i. Furthermore, it is already an upper bound on the response time for task τ_i. However, this bound is somewhat coarse and can be refined further by considering the release time of the job under analysis relative to the start of its busy window. This offset is customarily called A.

The aRSA framework establishes that the following condition implies a tighter bound R_i on the response time of any job $\tau_{i,j}$:

$$\begin{aligned} &\forall A.\ A < L_i \wedge \beta_i(A) \neq \beta_i(A+\epsilon) \implies \\ &\quad \exists F.\ A < F \leq A + R_i \\ &\qquad \wedge \max_0 \left(\{C_l - \epsilon \mid \tau_l \in \tau \wedge P_l < P_i\}\right) + \left(\sum_{\tau_h \in \tau \wedge P_i \leq P_h} RBF_h(F)\right) - (C_i - \epsilon) \leq SBF_i(F) \\ &\qquad \wedge SBF_i(F) + (C_i - \epsilon) \leq SBF_i(A + R_i) \end{aligned} \tag{3}$$

[1]

Intuitively, this condition defines a sparse *search space* $\{A \mid A < L_i \wedge \beta_i(A) \neq \beta_i(A+\epsilon)\}$ containing all "relevant" relative release times A with respect to the start of the busy window for any job $\tau_{i,j}$, and then takes the maximum over the response times for each relative release time. For any job $\tau_{i,j}$ with a relative release time A, F is the length of an interval such that by the end of it, the job $\tau_{i,j}$ is guaranteed to have started. To find such an F, we again follow the approach of defining an inequality between the maximum demand for processing time and the minimum guaranteed supply. Once the job $\tau_{i,j}$ starts, it can be delayed only by overheads, which are accounted for by SBF_i. Therefore, we can use SBF_i to find an interval $A + R$ such that the job has finished execution in this interval. Finally, the response-time bound R_i should be large enough such that it bounds the response-time for each possible relative release time.

Then, aRSA provides the following guarantee, conditional on providing a valid supply bound function SBF_i and proving that any Rössl schedule is work-conserving and priority-compliant (*i.e.,* assuming that the scheduler correctly implements a fixed-priority policy).

Theorem 1.1 (aRSA). *Any R_i that satisfies Eq. 3 is a response-time bound for τ_i.*

Practically speaking, finding a response-time bound R_i for each $\tau_i \in \tau$ then consists simply of solving Eq. 3 for each task, which is usually accomplished via fixed-point iteration for each A in the search space since Eq. 3 does not admit a closed-form solution.

1.2 Supply Bound Function

In the following, we define the supply bound function (SBF_i) that characterizes the worst-case impact of overheads in Rössl on the timely completion of jobs, as outlined in Section 4.4 of the paper. Note that, as briefly mentioned in the paper, the supply bound function gives a lower bound on the supply provided by the system in any busy window of a task under consideration.

[1] Recall that the busy windows are defined with respect to the release sequence as opposed to the original arrival sequence. Therefore, we use the propagated arrival curve instead of the original curve in this definition.

The aRSA framework's notion of "supply" is a general abstraction of any system resources allocated to executing jobs. For the concrete case of Rössl, we model the supply provided by the system at any time instant t as follows:

Definition 1.2 (Supply). For any time instant t and for any schedule *sched*, the supply at t, written $supply\_at(sched, t)$, is 1 if there exists some $\tau_{i,j}$ such that $sched\ t = Executes\ \tau_{i,j}$ or if $sched\ t = Idle$. For all other processor states, the supply provided by the processor is 0. The supply provided by the system in any interval $[t_1, t_2)$ is $supply\_in(sched, t_1, t_2) \triangleq \sum_{t=t_1}^{t<t_2} supply\_at(sched, t)$.

In other words, the supply in an interval $[t_1, t_2)$ models the system's capacity to do useful work, including any capacity idled away due to a lack of jobs to dispatch. In contrast, all overhead states (*e.g., PollingOvh*, *CompletionOvh*, *etc.*) are so-called *blackout states* in which the system does not provide any supply (*i.e.,* no job makes progress at those times).

The key property that our SBF has to satisfy is that it correctly lower-bounds the supply in any busy window $[t, t + \Delta)$ of a given task $\tau_i \in \tau$ under analysis:

$$\forall sched, \Delta, t.\ SBF_i(\Delta) \leq supply\_in(sched, t, t + \Delta) \tag{4}$$

In order to define SBF_i, we rely on an upper bound $BlackoutBound_i(\delta)$ on the blackouts (*i.e.,* times of no supply) in a busy window of length δ (w.r.t. task τ_i), which we define shortly below. Using this bound on blackouts, we state the SBF as follows:

$$SBF_i(\Delta) \triangleq \max_{\delta=0}^{\delta=\Delta}(\delta - BlackoutBound_i(\delta)) \tag{5}$$

The term $\delta - BlackoutBound_i(\delta)$ lower-bounds the supply that the system is guaranteed to provide in any busy window of length δ. We take the maximum over all interval lengths up to Δ in order to make the SBF monotonic: as increasing the length of the interval should not decrease the available total supply, aRSA requires SBFs to be monotonic. However, defining the SBF as $\Delta - BlackoutBound_i(\Delta)$ would violate monotonicity, as $BlackoutBound_i(\Delta)$ is not monotonic in Δ.

Defining the blackout bound. We split the task of bounding the total blackout in any busy window of a given task τ_i into: (1) defining a bound $TRB(\delta)$ on the blackouts caused by instances of *ReadOvh* and (2) defining a bound $NRB_i(\delta)$ on blackouts caused by instances of *PollingOvh*, *SelectionOvh*, *DispatchOvh*, and *CompletionOvh*. Thus, we define:

$$BlackoutBound_i(\delta) \triangleq NRB_i(\delta) + TRB(\delta) \tag{6}$$

At a high level, NRB_i relies on the fact that the bound needs to hold only inside a busy window of task τ_i and uses the bound on the number of job releases, the WCETs of all the blackout states (except for reading), and the scheduler protocol to compute a bound on non-reading overheads. *TRB* also depends on the scheduler protocol and the WCET of the reading state. However, since reads are driven by *job arrivals* as opposed to *job releases*, the *TRB* is not specific to τ_i and computed using the arrival curves of all tasks.

Bounding blackouts caused by reading. During the reading phase, Rössl iteratively attempts to read new packets from sockets until successive read attempts on all sockets have failed. Consequently, if new packets continuously arrive without pause, then theoretically Rössl's polling phase might never terminate (*i.e.,* the system might experience a case of "livelock"), to the effect of reads causing an unbounded amount of delay. Therefore, we first define a sufficient condition for the absence of nonterminating polling phases, and then define a bound on the total time spent in reading assuming this condition is satisfied.

Analogously to Eq. 2, the sufficient condition for the polling phase to terminate is formulated as an inequality on the arrival curve. In particular, our analysis requires the existence of a finite bound $RPB > 0$ that satisfies:

$$\left(\sum_{\tau_i \in \tau} \alpha_i \left(\max\left(PB + SB + DB + CB + (\max_{\tau_i \in \tau} C_i) + \epsilon, IB + \epsilon\right) + RPB + R\_Off\right)\right) \times RB \leq RPB. \quad (7)$$

Note that since reading is dependent only on the times at which jobs *arrive* as opposed to the times at which jobs are *released*, we use arrival curves when defining a bound on the time spent in reading.

Essentially, this inequality considers: (1) the maximum number of messages that could be waiting to be read in the system when the polling phase starts and (2) the maximum number of messages that arrive after the polling phase has started but that can be read during the interval and requires that *RPB* is an upper bound on the time spent reading these messages. The messages that arrive after the polling phase has started but that can be read during the interval can be easily computed using the term $\sum_{\tau_i \in \tau}(\alpha_i(RPB + R\_Off))$. [2] However, computing the maximum number of pending messages at the start of the polling phase is slightly more complicated and relies on the scheduler protocol.

To compute a bound on the maximum pending messages at the start of the polling phase, we proceed as follows. First, we know that at the beginning of any *PollingOvh* or *Idle* the set of read jobs is equal to the set of jobs that arrived so far. Let us refer to these points as "sync-points" (since the set of read jobs and the set of arrived jobs are in *synch* at these points). Next, we know from the scheduler protocol that a read has to be preceded either by *CompletionOvh* $\tau_{i,j}$ (for some $\tau_{i,j}$) or by *Idle*. In either case, we can obtain that a sync-point is a bounded distance away from the start of the polling phase. In particular, if the polling phase is preceded by a *CompletionOvh* then we know that in the interval $[t - (PB + SB + DB + CB + (\max_{\tau_i \in \tau} C_i) + \epsilon), t)$ there is at least one sync-point (in this case, the start of a *PollingOvh*). Instead, if the polling phase is preceded by an *Idle* then we know that in the interval $[t - (IB + \epsilon), t)$ there is at least one sync point (in this case, the start of an *Idle*).

Next, we show how, using the least positive solution *RPB* satisfying Eq. 7, we define the final bound on the maximum time spent by the system reading incoming packets in any interval of length δ.

$$TRB(\delta) \triangleq \sum_{\tau_i \in \tau} \alpha_i \left(RPB + \max\left(PB + SB + DB + CB + (\max_{\tau_i \in \tau} C_i) + \epsilon, IB + \epsilon\right) + \delta + R\_Off\right) \times RB \quad (8)$$

In this equation, the term $\delta + R\_Off$ bounds the time spent on reading jobs that arrive *during* the interval or within *R_Off* time after the interval has ended (and for which the reading phase could thus already have started within the interval). The remaining terms are upper-bounding the number of pending packets that are queued and waiting to be read at the start of the interval. To compute this bound, we have to consider the maximum pending reads that could ever have built up (and not just at the start of the polling phase as done above). Therefore, we have to consider every possible processor state that could be executing at the beginning of the interval under consideration.

Similar to the reasoning presented above, we bound the maximum pending reads by considering the arrivals since the closest sync-point. Then, bounding the pending reads boils down to bounding the time since that sync-point. This time is maximal at the end of any polling phase. Therefore, we have to bound the maximum number of pending messages by the maximum arrivals since the last sync-point before the end of the polling phase.

[2] The *R_Off* term shown above is a quirk of our abstract system model. Since a *ReadOvh* might be composed of up to *no_of_sockets* − 1 basic actions of type "Read ⊥" and one "Read" basic action, it might happen that a job arrives in the arrival sequence *after* the read for the job has already started. However, we can bound the time after which the job is guaranteed to have arrived into the system. This bound is represented by *R_Off*. We will define it in Def. 2.4.

Now, using the above equation, we know that, for any time t, if t is inside a polling phase then the polling phase must have started at most *RPB* time before t. We also know from above that a sync-point is at most $\max\left(PB + SB + DB + CB + (\max_{\tau_i \in \tau} C_i) + \epsilon, IB + \epsilon\right)$ time away from the start of the polling phase. Combining this we get that a sync point is at most

$$RPB + \max\left(PB + SB + DB + CB + (\max_{\tau_i \in \tau} C_i) + \epsilon, IB + \epsilon\right)$$

time away from the end of a polling phase. Therefore, this term is used to bound the pending reads as shown in the equation above.

Bounding blackouts caused by non-reading overhead states. Finally, we need to bound the delay caused by *PollingOvh*, *SelectionOvh*, *DispatchOvh*, and *CompletionOvh*. The flow property that we prove for Rössl (*i.e.,* the scheduler protocol) implies that each of these overheads happens once per execution of a job. Thus, we can state the bound on the non-reading blackout (NRB) as follows:

$$NRB_i(\delta) \triangleq \left(\sum_{\tau_j \in \tau \wedge P_i \leq P_j} \beta_j(\delta) \times (PB + SB + DB + CB)\right) + CB \tag{9}$$

Recall from §1.1 that busy windows are defined w.r.t. release curves. In Eq. 9, we thus use release curves (as opposed to arrival curves) since we need to consider all higher-and-equal-priority jobs executing in a busy window w.r.t. task τ_i of length δ. We further add one additional *CB* term to account for the possible completion of a lower-priority job that is released and starts execution *before* the start of τ_i's busy window.

2 ADEQUACY

In the following, we state our adequacy result including the full equation for the response-time bound.

Theorem 2.1 (Timing Correctness). *Assume a client P of Rössl according to Definition 3.4 in the paper, defining a set of tasks τ and the set of input sockets input_socks. Let the following be given:*

- *a set of arrival curves α_i (for each $\tau_i \in \tau$) which are valid (Def. 2.5),*
- *the WCETs for basic actions (WcetFR, WcetSR, WcetSel, WcetDisp, WcetCompl, WcetIdling) of which WcetSel, WcetDisp, WcetCompl and WcetIdling are strictly positive and $1 <$ WcetFR and $1 <$ WcetSR,*
- *the WCETs of callback executions C_i (for each task $\tau_i \in \tau$) that are strictly positive ($0 < C_i$).*

Furthermore, assume a run of Rössl as follows:

- *an execution $(P, \sigma_{init}) \rightarrow^{tr} (e_{final}, \sigma_{final})$ of P in the instrumented Caesium semantics starting in the initial state σ_{init} with trace tr*
- *arrival sequences arr_{sock} (for every $sock \in$ input_socks) which are valid (Def. 2.8),*
- *a list of timestamps ts, marking the start times for each marker function, which are valid for tr and arr (Def. 2.7)*
- *a horizon t_{hrzn} up to which the scheduler is known to have run that is valid (Def. 2.7)*

Then, we define an equation (Eq. 3) parametric in the arrival curve (α), the solutions of the two equations (Eq. 2) and (Eq. 7), the WCETs for the processor states (Def. 2.3), and the read offset (Def. 2.4).

Finally, for any task τ_i such that $\tau_i \in \tau$, if R_i is the solution for the response-time equation Eq. 3 for τ_i, then $R_i + \mathcal{J}_i$ (from Def. 4.3 in the paper) is a response-time bound (Def. 2.2) in tr and ts for the task τ_i.

Our final result is thus a response-time bound on the trace emitted by the operational semantics:

Definition 2.2 (Response-time bound on traces). R_i is a response time for the trace of marker functions tr, the timestamps ts, and the arrival sequence arr for the task $\tau_i \in \tau$ if:

$$\forall sock, j, t_{arr}.\ j \in arr_{sock}[t_{arr}] \wedge msg\_to\_task\ j = \tau_i \wedge t_{arr} + R_i < t_{hrzn} \implies$$
$$\exists k, t_{compl}. tr[k] = \mathsf{M\_Completion}\ j \wedge ts[k] = t_{compl} \wedge t_{compl} \leq t_{arr} + R_i$$

This definition essentially says that R_i is a response-time for a task τ_i if for every instance j of this task that arrives in the arrival sequence of any socket *sock* at some time t_{arr}, if j hasn't arrived late enough that executing the task will exceed the horizon t_{hrzn}, then there is a M_Completion j entry in the trace of marker functions *and* the timestamp for that entry is no greater than $t_{arr} + R_i$.

In the following we explain the assumptions of the theorem and define validity of the arrival curve and the timestamps.

WCETs. Since equations Eq. 7 and Eq. 8 which are used to define the *SBF* and, eventually, the response-time equation Eq. 3 are stated in terms of WCETs on processor states, we first define these WCETs in terms of the WCETs on basic actions that are given by the client as input:

Definition 2.3 (WCET of processor states).

$$\begin{aligned} RB &\triangleq WcetFR \times (length(input\_socks)) + WcetSR \\ PB &\triangleq WcetFR \times (length(input\_socks)) \\ SB &\triangleq WcetSel \\ DB &\triangleq WcetDisp \\ CB &\triangleq WcetCompl \\ IB &\triangleq WcetFR \times (length(input\_socks)) + WcetSel + WcetIdling \end{aligned}$$

Moreover, as our attribution of read sequences may lead to a job being read before it arrives, we define the maximum such offset as follows:

Definition 2.4 (Maximum interval between job arriving and being read).

$$R\_Off \triangleq WcetFR \times (length\ input\_socks - 1) + WcetSR$$

The arrival curve. Recall that an arrival curve α_i maps each message type τ_i and a duration length to the maximum number of arrivals of that message type in an interval of that length.

Definition 2.5 (Arrival curve validity). We say that a family of arrival curves is valid if:

- it is monotone for all message types: $\forall \tau_i \in \tau, \forall \Delta_1 \Delta_2, \Delta_1 \leq \Delta_2 \rightarrow \alpha_i(\Delta_1) \leq \alpha_i(\Delta_2)$
- the maximum number of arrivals for any message type for an interval of length 0 is 0: $\forall \tau_i \in \tau, \alpha_i(0) = 0$
- a solution to Eq. 2 exists for each task $\tau_i \in \tau s$ and a solution to Eq. (7) exists

Timestamps and the horizon.

Definition 2.6 (Timestamps respect WCETs). We say that a list of timestamps ts respect the WCETs for a trace of marker functions tr, if:

$$\forall i, j.\ tr[i] = \text{M\_Dispatch } j \implies ts[1+i]_? - ts[i] \leq WcetDisp \quad (10)$$

$$\forall i, j.\ tr[i] = \text{M\_ReadS} \land tr[1+i] = \text{M\_ReadE } sock\ j \implies ts[2+i]_? - ts[i] \leq WcetSR \quad (11)$$

$$\forall i, j.\ tr[i] = \text{M\_ReadS} \land tr[1+i] = \text{M\_ReadE } sock\ \bot \implies ts[2+i]_? - ts[i] \leq WcetFR \quad (12)$$

$$\forall i, j.\ tr[i] = \text{M\_ReadS} \land |tr| = i+1 \implies t_{hrzn} - ts[i] \leq min(WcetSR, WcetFR) \quad (13)$$

$$\forall i.\ tr[i] = \text{M\_Selection} \implies ts[1+i]_? - ts[i] \leq WcetSel \quad (14)$$

$$\forall i, j.\ tr[i] = \text{M\_Completion } j \implies ts[1+i]_? - ts[i] \leq WcetCompl \quad (15)$$

$$\forall i, k, j.\ tr[i] = \text{M\_Execution } j \land msg\_to\_task\ j = \tau_k \implies ts[1+i]_? - ts[i] \leq C_k \quad (16)$$

$$\forall i.\ tr[i] = \text{M\_Idling} \implies ts[1+i]_? - ts[i] \leq WcetIdling \quad (17)$$

where

$$ts[i]_? \triangleq i < |ts|\ ?\ ts[i]\ :\ t_{hrzn}$$

Definition 2.7 (Timestamps and horizon validity). We say that a horizon t_{hrzn} and a list of timestamps ts are valid for tr and arr, if:

- the timestamps are consistent with the assumed WCETs (Def. 2.6)
- the timestamps are monotonic with the horizon exceeding the last timestamp: $\forall i < length(ts).\ ts[i] < ts[i+1]_?$
- reads in tr can only appear in response to arrivals in the arrival sequence:
  - Each job is read only after it has arrived.

    $$\forall i, sock, j.\ tr[i] = \text{M\_ReadE } sock\ j \to \exists t_a.\ j \in arr_{sock}\ t_a \land t_a < ts[i]$$

  - If a read fails because no data is available, there are no unread arrived jobs.

    $$\begin{aligned} &\forall i, sock.\ tr[i] = \text{M\_ReadE } sock\ \bot \to \\ &\quad \forall j, t < ts[i].\ j \in arr_{sock}\ t \to j \in \text{read\_jobs } tr\ i \\ &\text{read\_jobs } tr\ i \triangleq \{j \mid \text{M\_ReadE } sock\ j \in tr[0:i]\} \end{aligned}$$

- the timestamp of the first basic action in the trace is bounded:
  $0 < ts[0]_? < (length(input\_socks)) \times WcetFR + WcetSel + WcetIdling$.
  Note that in the case of empty traces, this bound will be required to hold on t_{hrzn} instead.

Note that the bound on the first timestamp (or the horizon if log is empty) comes from the fact that the first timestamp should only be large enough to allow for n failed reads followed by a failed selection and the resulting idling. If the first timestamp is greater than that, then the WCET will be violated.

Arrival sequence.

Definition 2.8 (Arrival Sequence validity). We say that an arrival sequence arr is valid if:

- the job IDs in arrival sequence are unique:

  $$\forall sock_1, sock_2, t_1, t_2, j.\ j \in arr_{sock_1}\ t_1 \land j \in arr_{sock_2}\ t_2 \to sock_1 = sock_2 \land t_1 = t_2$$

- the total rate of all arrivals for any task ($\tau_i \in \tau s$) across all sockets is bounded by α_i:

$$\forall \Delta.\ |\{j | msg\_to\_task\ j = \tau_i \wedge \exists sock.\ j \in arr_{sock}\ t \wedge sock \in input\_socks \wedge t \in [t_1, t_1 + \Delta)\}| \leq \alpha_i(\Delta)$$

- all jobs come from some task set:

$$\forall sock, t, j.\ j \in arr_{sock}\ t \longrightarrow \exists \tau_i . msg\_to\_task\ j = \tau_i \wedge \tau_i \in \tau$$

REFERENCES

[1] Sergey Bozhko and Björn B Brandenburg. 2020. Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS)*. 22:1–22:24. https://doi.org/10.4230/DARTS.6.1.3