Real-time Systems

Carlson, Isovic, Hansson, Lundqvist, Nolte, Ouiment, Pettersson, Punnekkat, Seceleanu
Mälardalen Real-Time Research Centre
School of Innovation, Design & Engineering, Mälardalen University
2009

Errata:

Page 62: “Figure 3.1 a)” should be “Figure 3.2 a)”

Page 90: The semaphore priority ceilings ceil(S2) = prio(t2) = 2 should be 3.

Page 98: D should be {4, 7, 10, 15, 16, 22, 23 }

Page 129: In Example 4.5.15, C = 75 should be 85

Page 152: “Section 7.2” should be “Section 5.2”

- τ_i – periodic task *i* (Greek symbol τ is pronounced as "tau")
- τ_i^j – the *j*th invocation of τ_i

Parameters that characterize a task τ_i:

- T_i – period time of the task
- D_i – relative deadline of the task
- $WCET_i$ – worst-case execution time of the task
- O_i – offset of the task
- R_i – response time (the longest possible) of the task

Parameters that characterize a task instance τ_i^j :

- a_i^j – arrival time of the instance
- st_i^j – start time of the instance
- ft_i^j – finishing time of the instance
- c_i^j – actual execution time of the instance
- r_i^j – actual response time of the instance
- d_i^j – absolute deadline of the instance

Now it gets easier to express the dependencies between the parameters, since we can use mathematic expressions instead of words. For example, the relation between the absolute and relative deadline can be expressed as:

$$d_i^j = a_i^j + D_i$$

In the same way, the actual response time of a task instance can be obtained as:

$$r_i^j = ft_i^j - a_i^j$$

The worst-case response time of a task, R_i, is the maximum of all response times of all individual task instances. We will see later in this chapter how R_i is calculated.

Task types

The most common task type in real-time systems are periodic tasks, but there are some other types as well, such as aperiodic and sporadic tasks. Here we describe them in terms of their parameters.

Periodic tasks – A periodic task τ_i can be described by four parameters (offset, execution time, deadline, and period time):

$$\tau_i = \{O_i, C_i, D_i, T_i\}$$

As mentioned before in Chapter 2, a periodic task consists of an infinite sequence of identical instances or jobs that are activated within regular time periods, which are calculated as:

$$a_i^0 = O_i$$

$$\forall j > 0, a_i^j = a_i^{j-1} + T_i$$

Another way to express the arrival times of the instances is (except the first one, which is equal to offset):

$$\forall j > 0, a_i^j = O_i + (j-1)T_i$$

If there is no offset specified, the task instances will be activated at times *0*, T_i, $2T_i$, $3T_i$, ...

Sporadic tasks – This type of tasks is used to handle events that arrive at the system at arbitrary points in time, but with defined maximum frequency. Just like periodic tasks, they are invoked repeatedly with a (non-zero) lower bound on the duration between consecutive invocations, but the difference is that a sporadic task may invoke its instances irregularly. Before run-time, it is known what the minimum time between consecutive instances is, which is called **minimum inter-arrival time** (T^{min}), but the actual time between arrivals of instances is not know until run-time, i.e., it first becomes known when the instance arrives.

A sporadic task is usually expressed with three parameters (execution time, deadline, and minimum inter-arrival time):

$$\tau_i = \left\{C_i, D_i, T_i^{min}\right\}$$

The following must hold for all instances of a sporadic task:

$$\forall j > 0, a_i^j \geq a_i^{j-1} + T_i^{\min}$$

$$\forall j > 0, d_i^j = a_i^j + D_i$$

Note "greater or equal" for arrival times, which means that the next instance can be invoked the earliest after T^{min} time units, counted from the arrival of the current instance. The exact arrival times are not known (until they actually occur at run-time).

Note the difference to periodic tasks, where we know before run-time that the instances will be invoked with exactly *T* time units in between. Periodic tasks are activated with regular periodicity by the system clock, while sporadic tasks usually wait for some event, which in general is not periodic (e.g., arrival of a data packet from a network).

The term I_i, called **interference**, is the preemption time from higher-priority tasks. For the highest-priority task in the system, the response time will be equal to its own execution time, $R_i=C_i$, since no other tasks will preempt it. Other tasks will suffer interference from higher-priority tasks. The problem now becomes finding the interference time for a task τ_i.

Let τ_k be a task with higher priority than τ_i, which is released at the same time as τ_i. i.e., at time t. Task τ_i will have an instance that becomes ready at t and finishes its execution at time $t+R_i$ (we assume this is the instance with the worst-case response time). During its execution, the instance of τ_i will be pre-empted by the higher-priority task τ_k. If τ_k has a shorter period than τ_i, then it will preempt τ_i several times, i.e., several instances of τ_k will occur and preempt the current instance of τ_i.

The number of instances of τ_k that occur in the interval $[t, t+R_i]$, and hence interfere with (preempt) τ_i can be calculated by dividing the length of the interference interval by the activation frequency of τ_k, i.e., its period time:

$$\left\lceil \frac{R_i}{T_k} \right\rceil$$

The symbol $\lceil \square \rceil$ is the ceiling function, and it is a round-up function. So, for example $\lceil 2.3 \rceil = 3$.

The total time taken by task τ_k when it preempts and executes is simply the number of instances of τ_k, calculated as above, multiplied by its execution time, C_k:

$$\left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

So, for the total interference term, we simply add this up for all the tasks with higher priority than τ_i:

$$I_i = \sum_{\forall k \in hp(\tau_i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

where $hp(\tau_i)$ is the set of tasks that have higher priority than task τ_i .

Hence, the worst-case response time for a task τ_i is given by:

$$R_i = C_i + \sum_{\forall k \in hp(\tau_i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Since R_i is represented on both sides of the equation, it can be solved by forming a recurrence relation, i.e., the next value is obtained based on the currently calculated value:

$$R_i^{n+1} = C_i + \sum_{\forall k \in hp(\tau_i)} \left\lceil \frac{R_i^n}{T_k} \right\rceil C_k \qquad (5)$$

The initial value of *Ri* is the execution time *Ci*, since the shortest possible response time of τ_i is its worst-case execution time. The set of consecutive values {*R0*, *R1*, *R2*,...} is monotonically non-decreasing, i.e., the next value is greater than or equal to the previously calculated value, and the sequence will converge to the smallest value of *Ri* that satisfies equation (5).

Simply said, we stop iterating when $R_i^{n+1} = R_i^n$, and if *Ri* ≤ *Di*, we conclude that the task is schedulable. Otherwise, if some of the iteration steps result in a value of *Ri* that is larger than *Di*, we should stop with the iterations and conclude that the task is not schedulable.

Example: Is the following task set schedulable by Rate Monotonic?

Task	C_i	$T_i=D_i$
τ_1	1	3
τ_2	1	6
τ_3	1	5
τ_4	2	10

An observant reader will notice that this task set is the same as the one presented in Section 3.8, where we explained the Rate Monotonic algorithm. We calculated before that *U=0.9* is larger than the upper bound 0.75. Hence, the utilization test could not give us an answer to whether the task set is schedulable or not. It was only after running and analyzing the execution trace that we could see that the set was schedulable.

Instead of analyzing the execution trace, we can use Response Time Analysis to come to the same conclusion **before run-time**. We use equation (5) to calculate the response times for all tasks and compare them to their deadlines.

<u>Response time of task τ_1:</u>

According to Rate Monotonic, task τ_1 is assigned the highest priority. The set of high-priority tasks *hp*(τ_1) is empty; hence, the response time of τ_1 will be equal to its execution time:

$$hp(\tau_1) = \{\} \Rightarrow R_1 = C_1 = 1 \leq D_1$$

<u>Response time of task τ_2:</u>

Task τ_2 will be assigned higher priority than τ_4, but lower priority than τ_1 and τ_3, which both have shorter periods. Hence, both τ_1 and τ_3 will influence the response time of τ_2.

$hp(\tau_2) = \{\tau_1, \tau_3\}$

$R_2^0 = C_2 = 1$

$$R_2^1 = C_2 + \left\lceil \frac{R_2^0}{T_1} \right\rceil C_1 + \left\lceil \frac{R_2^0}{T_3} \right\rceil C_3 = 1 + \left\lceil \frac{1}{3} \right\rceil 1 + \left\lceil \frac{1}{5} \right\rceil 1 = 1 + 1 + 1 = 3$$

$$R_2^2 = C_2 + \left\lceil \frac{R_2^1}{T_1} \right\rceil C_1 + \left\lceil \frac{R_2^1}{T_3} \right\rceil C_3 = 1 + \left\lceil \frac{3}{3} \right\rceil 1 + \left\lceil \frac{3}{5} \right\rceil 1 = 1 + 1 + 1 = 3$$

Since $R_2^2 = R_2^1, we\ can\ stop\ iterations. Hence\ R_2 = 3 \leq D_2$

<u>Response time of task τ_3:</u>

$hp(\tau_3) = \{\tau_1\}$

$R_3^0 = C_3 = 1$

$$R_3^1 = C_3 + \left\lceil \frac{R_3^0}{T_1} \right\rceil C_1 = 1 + \left\lceil \frac{1}{3} \right\rceil 1 = 1 + 1 = 2$$

$$R_3^2 = C_3 + \left\lceil \frac{R_3^1}{T_1} \right\rceil C_1 = 1 + \left\lceil \frac{2}{3} \right\rceil 1 = 1 + 1 = 2$$

$R_3^2 = R_3^1 \Rightarrow R_3 = 2 \leq D_3$

<u>Response time of task τ_4:</u>

$hp(\tau_4) = \{\tau_1, \tau_2, \tau_3\}$

$R_4^0 = C_4 = 2$

$$R_4^1 = C_4 + \left\lceil \frac{R_4^0}{T_1} \right\rceil C_1 + \left\lceil \frac{R_4^0}{T_2} \right\rceil C_2 + \left\lceil \frac{R_4^0}{T_3} \right\rceil C_3 = 2 + \left\lceil \frac{2}{3} \right\rceil 1 + \left\lceil \frac{2}{6} \right\rceil 1 + \left\lceil \frac{1}{5} \right\rceil 1 = 5$$

$$R_4^2 = C_4 + \left\lceil \frac{R_4^1}{T_1} \right\rceil C_1 + \left\lceil \frac{R_4^1}{T_2} \right\rceil C_2 + \left\lceil \frac{R_4^1}{T_3} \right\rceil C_3 = 2 + \left\lceil \frac{5}{3} \right\rceil 1 + \left\lceil \frac{5}{6} \right\rceil 1 + \left\lceil \frac{5}{5} \right\rceil 1 = 6$$

$$R_4^3 = C_4 + \left\lceil \frac{R_4^2}{T_1} \right\rceil C_1 + \left\lceil \frac{R_4^3}{T_2} \right\rceil C_2 + \left\lceil \frac{R_4^3}{T_3} \right\rceil C_3 = 2 + \left\lceil \frac{6}{3} \right\rceil 1 + \left\lceil \frac{6}{6} \right\rceil 1 + \left\lceil \frac{6}{5} \right\rceil 1 = 7$$

$$R_4^4 = C_4 + \left\lceil \frac{R_4^3}{T_1} \right\rceil C_1 + \left\lceil \frac{R_4^3}{T_2} \right\rceil C_2 + \left\lceil \frac{R_4^3}{T_3} \right\rceil C_3 = 2 + \left\lceil \frac{7}{3} \right\rceil 1 + \left\lceil \frac{7}{6} \right\rceil 1 + \left\lceil \frac{7}{5} \right\rceil 1 = 9$$

$$R_4^5 = C_4 + \left\lceil \frac{R_4^4}{T_1} \right\rceil C_1 + \left\lceil \frac{R_4^4}{T_2} \right\rceil C_2 + \left\lceil \frac{R_4^4}{T_3} \right\rceil C_3 = 2 + \left\lceil \frac{9}{3} \right\rceil 1 + \left\lceil \frac{9}{6} \right\rceil 1 + \left\lceil \frac{9}{5} \right\rceil 1 = 9$$

$R_4^5 = R_4^4 \Rightarrow R_4 = 9 \leq D_4$

worst-case timing performance of the ICPP protocol is the same as PCP, the analysis developed for PCP (i.e., the calculation of blocking factors) remains unchanged for ICPP.

Response Time Analysis with blocking

We will now see how we can extend the response time analysis in order to include the effects of blocking. As we showed above, a task τ_i can be blocked by lower-priority tasks due to shared resources. The delay caused by blocking is called the blocking factor, and for task τ_i is denoted as B_i. It is a function of the length of the critical sections of the lower-priority tasks that can block. In other words, the **blocking factor** of a task τ_i is the longest time a task can be delayed by the execution of lower-priority tasks.

In PCP, a given task τ_i is blocked by at most **one** critical section of any lower-priority task, among all tasks that can lock a semaphore with a priority ceiling greater than or equal to the priority of task τ_i.

What this means is: First, we identify all the tasks with lower priority than task τ_i. Second, we identify all the semaphores that the lower-priority tasks can lock. Third, we select from these only those semaphores where the ceiling of the semaphore has a priority higher than or the same as the priority of task τ_i. Finally, we look at the computation time duration that each lower-priority task is holding its semaphores (critical section), and the longest of these computation times is the blocking factor, B_i.

Once when we calculated the blocking factor, it is easily added to the response time analysis equation:

$$R_i^{n+1} = C_i + B_i + \sum_{\forall k \in hp(\tau_i)} \left\lceil \frac{R_i^n}{T_k} \right\rceil C_k \tag{6}$$

Example: Is the following task set schedulable by Rate Monotonic if Priority Ceiling Protocol is used for shared resources?

τ_i	C_i	T_i=D_i	S_k	$cs(\tau_i,S_k)$
τ_1	10	100	S_1	1
τ_2	12	40	S_1	2
			S_2	1
τ_3	6	50	S_1	1

As we can see in the example above, EDF will always give the highest priority to the task whose current instance has the shortest deadline. As a consequence, different task instances from the same task might have different priority, depending on the other tasks that are ready at the moment. Therefore, we say that the priority assignment is dynamic. In Rate Monotonic, on the other hand, all instances of the same task will have the same priority and will not be changed at runtime (unless we use some of the semaphore protocols explained above).

Processor Demand Analysis

Processor utilization analysis for EDF is valid only if deadline is equal to period for all tasks in a task set. If this is not the case, i.e., if deadline is less than period, **Processor Demand Analysis** (PDA) can be used.

The processor demand for a task τ_i in a given time interval [0, L] is the amount of processor time that the task needs in the interval in order to meet the deadlines that fall within the interval. Let N_i represent the number of instances of τ_i that must complete execution before L. We can calculate N_i by counting how many times task τ_i has been released during the interval [0, L-D_i], i.e., N_i can be expressed as:

$$N_i = \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1$$

The total processor demand for all tasks in the interval is thus:

$$C^T(0, L) = \sum_{i=1}^{n} N_i C_i = \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

Finally, a sufficient and necessary condition for EDF when deadlines are less than periods is given by:

$$\forall L \epsilon D : L \geq \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \qquad (8)$$

$$D = \{ d_i^j \mid d_i^j = jT_i + D_i, d_i^j < LCM, 1 \leq i \leq n, j \geq 0 \}$$

i.e., we check intervals from 0 to each of the deadlines, until we either have checked intervals up to the LCM or the condition fails for some deadline.

Example: Is the following task set schedulable by EDF?

Task	C_i	D_i	T_i
τ_1	3	4	6
τ_2	4	7	8

Deadlines up to the LCM:

D = {4, 7, 10, 15, 16, 22, 23}

$$C^T(0,7) = \sum_{i=1}^{n}\left(\left\lfloor\frac{7-D_i}{T_i}\right\rfloor + 1\right)C_i = \left(\left\lfloor\frac{7-4}{6}\right\rfloor + 1\right)3 + \left(\left\lfloor\frac{7-7}{8}\right\rfloor + 1\right)4 = 7 \leq 7$$

$$C^T(0,10) = \left(\left\lfloor\frac{10-4}{6}\right\rfloor + 1\right)3 + \left(\left\lfloor\frac{10-7}{8}\right\rfloor + 1\right)4 = 10 \leq 10$$

$$C^T(0,15) = \left(\left\lfloor\frac{15-4}{6}\right\rfloor + 1\right)3 + \left(\left\lfloor\frac{15-7}{8}\right\rfloor + 1\right)4 = 14 \leq 10$$

$$C^T(0,16) = \left(\left\lfloor\frac{16-4}{6}\right\rfloor + 1\right)3 + \left(\left\lfloor\frac{16-7}{8}\right\rfloor + 1\right)4 = 17 > 16 \Rightarrow \textit{Deadline missed!}$$

Answer: The set is not schedulable; a deadline will be missed at time 16.

3.13 Comparison between Rate Monotonic and Earliest Deadline First

We will conclude this section about online scheduling by presenting a short comparison between EDF and RM, based on some selected criteria.

- RM and EDF have the same implementation complexity – A small additional overhead is needed in EDF to update the absolute deadline at each instance release.
- RM is supported by commercial RTOSs – One big advantage of RM is that it can be easily implemented on top of fixed priority kernels.
- Run-time overhead is smaller in EDF – Due to the smaller number of context switches.
- EDF utilizes the processor better than RM – EDF achieves full processor utilization, 100%, whereas RM only guarantees 69%.
- EDF is simpler to analyze if *D* = *T* – This is important for reducing admission control overhead in small embedded systems.
- Jitter reduction – EDF is fair in reducing jitter, whereas RM only reduces the jitter of the highest-priority tasks.
- Aperiodic task handling – EDF is more efficient than RM for handling aperiodic tasks.

4.5.13 Sufficient response-time test

The worst-case response time of a frame is found in the busy period beginning with a **critical instant**, where all frames of priority *i* and higher are simultaneously queued at their corresponding communication adapters. Following this, the worst-case response time for frame *i* is given by

$$R_i = J_i + w_i + C_i \tag{3}$$

where J_i is the queuing jitter of frame *i*, i.e., the maximum variation in queuing time relative to the start of the frame period T_i, inherited from the sender task that queues the frame, C_i is the worst-case transmission time of frame *i* (given by Equation (2)), and w_i is the queuing delay given by solving the equation

$$w_i^{n+1} = B^{MAX} + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{4}$$

where B^{MAX} corresponds to the transmission time of the longest possible CAN frame (i.e., the worst-case transmission time of a CAN frame with 8 bytes of payload data), $hp(i)$ is the set of frames with priority higher than that of frame *i*. Note that Equation (4) is a recurrence relation with an initial value that can be chosen equal to C_i, i.e., $w_i^0 = C_i$, and a terminating condition of either $w_i^{n+1} = w_i^n$ and/or $J_i + w_i^{n+1} + C_i > D_i$. Only if the latter condition is false is the frame schedulable.

4.5.14 Exact response-time test

For many applications, it is sufficient to use the slightly pessimistic response-time calculation given by R_i in Equation (3). However, the exact worst-case response time $\overline{R_i}$ of a CAN frame *i* sent on stream S_i is found within what is called the level-*i* busy period. Specifically, the individual frame response time has to be calculated for **each instance** of frame *i* within its busy period. The worst-case response time $\overline{R_i}$ is experienced by one or more of the instances of a specific frame within its corresponding busy period. Hence, in order to derive the exact worst-case response time, the response time has to be calculated for all these frame instances.

As a first step, the length t_i of the level-*i* busy period is given by solving the following recurrence relation, starting with an initial value of $t_i^0 = C_i$, and finishing when $t_i^{n+1} = t_i^n$

$$t_i^{n+1} = B_i + \sum_{\forall k \in hep(i)} \left\lceil \frac{t_i^n + J_k}{T_k} \right\rceil C_k \tag{5}$$

where B_i is the maximum blocking time due to lower-priority frames in the process of frame transmission, $hep(i)$ is the set of frames with priority higher than or equal to that of frame *i*, J_k is the queuing jitter of frame *k*, i.e., the maximum variation in queuing time relative to the start of the frame period T_k, inherited from the sender task that queues the frame, and C_k is the frame transmission time for frame *k* derived from Equation (2).

Now, looking at a specific frame *i*, the number of instances Q_i of frame *i* that become ready for frame transmission before the end of the busy period is given by

$$Q_i = \left\lceil \frac{t_i + J_i}{T_i} \right\rceil \qquad (6)$$

For each instance ($0 \cdots Q_i - 1$) of frame *i*, the corresponding worst-case frame response time must be derived. Letting *q* be the index of a frame instance, the worst-case response time $\overline{R_i}(q)$ of frame instance number *q* is given by

$$\overline{R_i}(q) = J_i + \overline{w_i}(q) - qT_i + C_i \qquad (7)$$

where $\overline{w_i}(q)$ represents the effective queuing time, given by the recurrence relation in Equation (8), starting with an initial value of $\overline{w}_i^0(q) = 0$, and finishing when $\overline{w}_i^{n+1}(q) = \overline{w}_i^n$ (q) or when $J_i + \overline{w}_i^{n+1}(q) - qT_i + C_i > D_i$ (i.e., either when a worst-case response time is found, or when the frame is found to be not schedulable). In Equation (8), τ_{bit} is the bit time.

$$\overline{w}_i^{n+1}(q) = B_i + qC_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{\overline{w}_i^n(q) + J_j + \tau_{bit}}{T_j} \right\rceil C_j \qquad (8)$$

Once all Q_i worst-case response times are calculated, the worst-case response time $\overline{R}_i$ for frame *i* is found as the maximum response time among these Q_i instances' response times as follows:

$$\overline{R}_i = \max_{q=0 \cdots Q_i-1} (\overline{R}_i(q)) \qquad (9)$$

Note that this exact analysis is also valid for frames with deadlines greater than their period, which is not the case for the sufficient response-time test outlined above.

4.5.15 Example

Here, an example of the above analyses is given in order to illustrate their differences. More precisely, a set S containing 3 frames will be deemed not schedulable using the sufficient response-time test, and schedulable using the exact response-time test.

Consider the frame set outlined in Table 1. The basic assumptions on the system are that the bus speed is 1 Mbps, i.e., $\tau_{bit} = 1\mu s$, and there is no jitter, i.e., $J = 0$ for all frames. All frames have *2* bytes of payload data, hence, according to Equation (2), $C = 75$. The bus utilization is rather high; ~97%. The reason for having such a high bus utilization is to force a delicate scenario not suitable for the sufficient analysis, but captured by the exact analysis.

Table 4.1: Frame set under analysis

Frame *i*	P_i	T_i	D_i	C_i
1	1	187.5	187.5	75
2	2	262.5	262.5	75
3	3	262.5	262.5	75

4.5.16 Sufficient response-time test

Starting with the sufficient response-time test, using Equation (4) to calculate w_i together with Equation (3) to calculate R_i for each frame *i* gives the following (recall that Equation (4) is a recurrence equation terminating when $w_i^{n+1} = w_i^n$):

$$w_1^0 = 0, w_1^1 = 75, w_1^2 = \mathbf{75}$$

$$R_1 = \mathbf{150}$$

$$w_2^0 = 0, w_2^1 = 150, w_2^2 = \mathbf{150}$$

$$R_2 = \mathbf{225}$$

$$w_3^0 = 0, w_3^1 = 225, w_3^2 = 300, w_3^3 = 375, w_3^4 = 450, w_3^5 = \mathbf{450}$$

$$R_3 = \mathbf{525}$$

The above results say that frame 1 and 2 are schedulable, i.e., $R_1 \leq D_1$ and $R_2 \leq D_2$. However, $R_3 > D_3$ indicate that a deadline is missed. Hence, the frame set S is deemed not schedulable.

4.5.17 Exact response-time test

On the other hand, if exact analysis is used, the above frame set S is, in fact, schedulable. To show this, Equations (5) - (9) are used to calculate the frame response times for all 3 frames.

As a first step, using Equation (5), the level-*i* busy period has to be calculated for each frame to see, using Equation (6), if multiple instances of that frame are present within the busy period. If that is the case, the response time has to be calculated for all these frame instances using Equations (7) - (8), and the maximum response time among all instances has to be selected using Equation (9).

Starting with frame 1, using Equation (5), the level-1 busy period is calculated to be $t_1^0 = 75, t_1^1 = 150, t_1^2 = \mathbf{150}$. Now, $Q_1 = 1$ according to Equation (6). Hence, there is only once instance of frame 1 within the level-1 busy period. For this instance, the response time is calculated using Equation (7) - (8):

$\overline{w}_1^0(0) = 0, \overline{w}_1^1(0) = 75, \overline{w}_1^2(0) = 75$

$R_1(0) = 150$

For frame 2, the same procedure is repeated using Equation (5) giving $t_2^0 = 75, t_2^1 = 225, t_2^2 = 300, t_2^3 = 375, t_2^4 = 450, t_2^5 = 450$, hence $Q_2 = 2$ according to Equation (6), i.e., there are two instances of frame 2 in the level-2 busy period. Here, the response time is calculated for both these instances using Equations (7) - (8), and the maximum is selected using Equation (9):

$\overline{w}_2^0(0) = 0, \overline{w}_2^1(0) = 150, \overline{w}_2^2(0) = 150$

$R_2(0) = 225$

$\overline{w}_2^0(1) = 0, \overline{w}_2^1(1) = 225, \overline{w}_2^2(1) = 300, \overline{w}_2^3(1) = 300$

$R_2(1) = 112.5$

$R_2 = 225$

Finally, for frame 3, there are 2 instances within the level-3 busy period as $t_3^0 = 75, t_3^1 = 225, t_3^2 = 300, t_3^3 = 450, t_3^4 = 525, t_3^5 = 525$ and $Q_3 = 2$. Following the same reasoning as for frame 2, the response time is calculated:

$\overline{w}_3^0(0) = 0, \overline{w}_3^1(0) = 150, \overline{w}_3^2(0) = 150$

$R_3(0) = 225$

$\overline{w}_3^0(1) = 0, \overline{w}_3^1(1) = 225, \overline{w}_3^2(1) = 300, \overline{w}_3^3(1) = 375, \overline{w}_3^4(1) = 450, \overline{w}_3^5(1) = 450$

$R_3(1) = 262.5$

$R_3 = 262.5$

To summarize, using exact analysis rather than sufficient analysis, the whole frame set is schedulable as $\overline{R}_1 \leq D_1$, $\overline{R}_2 \leq D_2$ and $\overline{R}_3 \leq D_3$. The sufficient test is much faster, as it requires fewer calculations to be performed. In most cases, the sufficient analysis will yield the same frame sets as schedulable as the exact analysis will. However, there are cases, as shown in the example above, where only the exact analysis will show that a set of frames is, in fact, schedulable.

4.5.18 Holistic analysis

In this section, we show a way of how to apply the CAN analysis presented above in a distributed system comprised of **end-to-end** timing requirements. We assume that a distributed system consists of a number of nodes that are interconnected with a CAN bus. On the nodes, a number of tasks are executing, scheduled by a preemptive fixed-priority real-time scheduler. Timing requirements can be both