Bounding the End-to-End Response Times of Tasks in a Distributed Real-Time System Using the Direct Synchronization Protocol

Jun Sun Jane Liu

Abstract

In a distributed real-time system, a task may consist of a chain of subtasks which execute on different processors. In order to guarantee that all timing constraints are met in such a system, it is imperative to be able to determine the end-to-end response time for each task. This report focuses on distributed systems where tasks are periodic and scheduled according to fixed priority scheduling algorithms. Moreover, if a subtask has predecessors, every instance of the subtask is released as soon as the corresponding instance of the immediate predecessor subtask completes. This report describes an algorithm that can be used to find upper bounds on the end-to-end response times of tasks in such a distributed real-time system.

1 Assumptions and Notations

In a distributed real-time system, a task may consist of a chain of subtasks. Subtasks in the same task may execute on different processors and have different priorities. On each processor, subtasks are scheduled according to their assigned fixed priorities. One example is shown in Figure 1. There are four processors and three tasks in the system. Task T_1 has only one subtask, which is the trivial case. Task T_2 executes on processor P_4, P_2, P_1 and P_3 sequentially. It has four subtasks. Task T_3 executes on processor P_3 first, then on P_4 and then back to P_3 again. It has three subtasks. In general, an *end-to-end task* consists of a chain of subtasks which execute on different processors, and the task has a single deadline which specifies the maximum allowed duration from the release of its first subtask till the completion of its last subtask. The deadline for an end-to-end task is often called the *end-to-end deadline*.

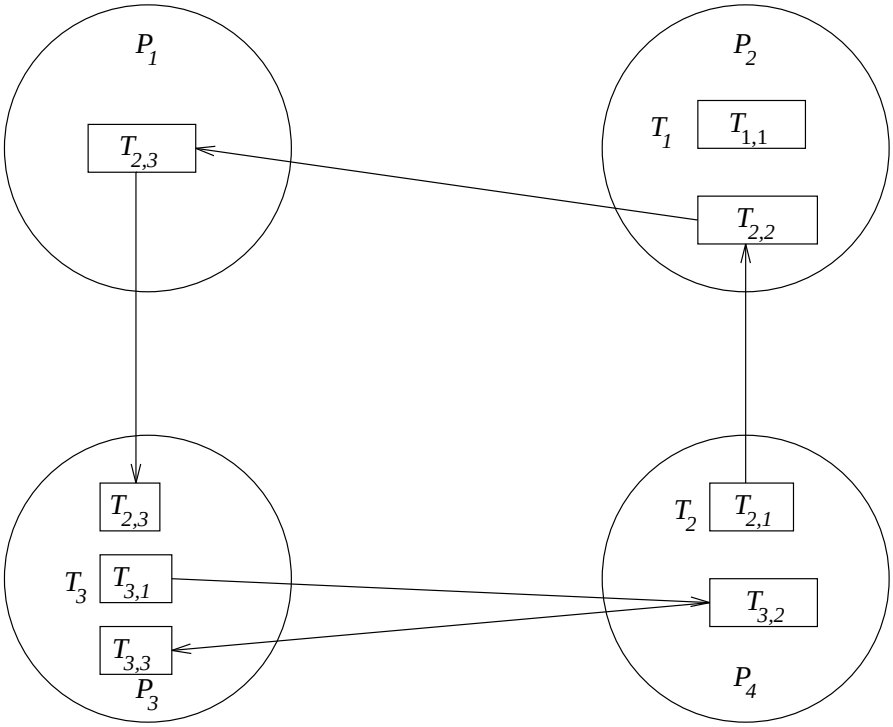

Figure 1: An End-to-End System

In this report we focus our attention on tasks that are periodic. By a task being periodic, we mean that the first subtask of the task is released periodically and later subtasks in the chain are released according to the *synchronization protocol* used in the system. (More precisely speaking, the inter-arrival time between two consecutive instances of the first subtask is no shorter than the period.) This report considers only the *Direct Synchronization (DS)* protocol. According to this protocol, an instance of a subtask that has predecessors is released immediately after the corresponding instance of its immediate predecessor completes. (For more discussion on synchronization protocols, interested readers can refer to [1].) For example, a *monitoring* task in a control system may have three subtasks: (1) the *sampling* subtask which samples physical parameters on the field processor, (2) the *transmission* subtask which transmits the date over a communication link, and (3) the *displaying* subtask which processes the data and displays them on the control processor. If the DS protocol is used in the system, the *transmission* subtask will send the data immediately when the data are obtained by the *sampling* subtask, and the *displaying* subtask will start to process the data as soon as the data arrive at the control processor.

The *end-to-end response (EER) time* of a task (instance) is the length of time from the release of an instance of its first subtask till the completion of the corresponding instance of its last subtask. In this report, we address the problem of how to compute an upper bound on the end-to-end response times of tasks in a distributed system when the execution of subtasks is synchronized according to the DS protocol. The assumptions we make in this report are listed below.

1. The system consists of a set $\{P_i\}$ of processors and a set $\{T_i\}$ of end-to-end tasks.

2. An end-to-end task T_i consists of a chain of subtasks, $T_{i,1}$, $T_{i,2}$, ..., T_{i,n_i}. There are precedence constraints between any two consecutive subtasks in the same chain. Specifically, an instance of subtask $T_{i,j}$ must complete before the corresponding instance of $T_{i,j+1}$ can start.

3. Each task T_i is a periodic task. The period of T_i is p_i. Again, we mean that the first subtask ($T_{i,1}$) is released periodically at the period p_i and later subtasks in the chain are released according to the DS protocol.

4. The maximum execution time of each subtask $T_{i,j}$ is $\tau_{i,j}$. Furthermore, the minimum execution time of each subtask is greater than zero.

5. Each subtask is assigned a fixed priority. Subtasks on each processor are scheduled together based on their priorities.

6. If there are multiple instances of $T_{i,j}$ are ready for execution, they are scheduled on a FIFO basis. Let $T_{i,j}(g)$ denote the gth instance of $T_{i,j}$. It is equivalent to say that subtask instance $T_{i,j}(g)$ must complete before $T_{i,j}(g+1)$ can start. This is the strategy typically used in practice.

A closely related work is done by Tindell, et al. [2], where a similar problem is addressed. Near the end of this report, we will compare our approach with the one in [2].

2 Clumping Effect

In a system that uses the DS protocol, the *time demand analysis* for periodical task systems [3, 4, 5] cannot be used directly to obtain a bound on the response time of each subtask. In the worst

case, several instances of a higher-priority subtask may execute back-to-back consecutively or rather closely together in time, a phenomenon called the *clumping effect*. The example in Figure 2 illustrates this effect. Each circle in the picture represents a processor, and each box represents a subtask that executes on the processor. The subtask parameters are listed in the parentheses in each box representing the subtask. The first number is the period, and the second is the maximum execution time. The end-to-end relative deadline for each task is equal to its period.

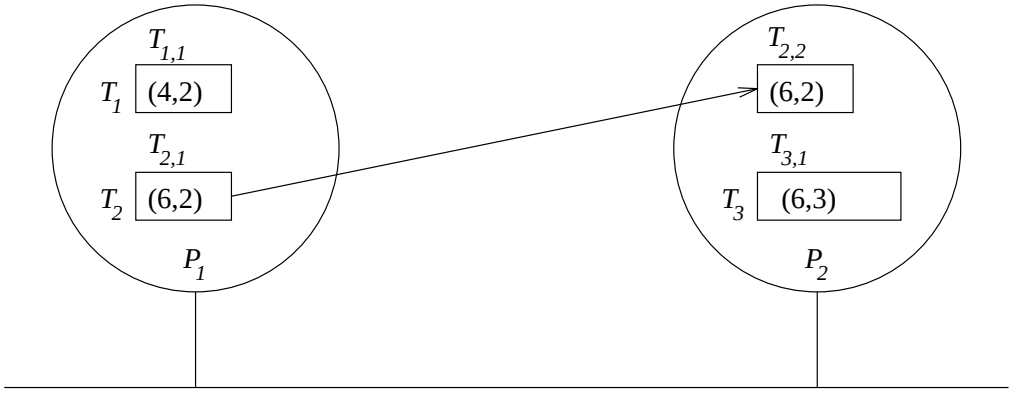

Figure 2: Example to Illustrate the Clumping Effect

We assume that on P_1 subtask $T_{1,1}$ has higher priority than $T_{2,1}$, and on P_2 subtask $T_{2,2}$ has higher priority than $T_{3,1}$. In addition, we assume that task T_3 has a phase of 4 and others have zero phases. A schedule for the first 10 time units of this simple system is shown in Figure 3. In the schedule, we notice that the first two instances of $T_{2,1}$ complete at time 4 and 8, causing two instances of $T_{2,2}$ to be released at those times. Because the completion times of $T_{2,1}$ are not periodic, the release times of $T_{2,2}$ are not periodic. As a result, the first two instances of $T_{2,2}$ execute close together, which causes the first instance of $T_{3,1}$ to be preempted twice by $T_{2,2}$, and T_3 misses its deadline at time 10. On the other hand, if $T_{2,2}$ were released periodically, $T_{3,1}$ would never be preempted twice by $T_{2,2}$ because they have the same period.

In general, the completion time of a subtask can vary widely, especially if the subtask is near the tail of a long subtask chain. The variation is not only caused by preemption by higher priority subtasks but also due to variations in the execution times of subtasks. If one instance of $T_{i,j}$ completes rather late but all subsequent instances complete relatively early, it is conceivable that many instances

of $T_{i,j+1}$ can be released and ready for execution almost at the same time. If there is another subtask on the same processor as $T_{i,j+1}$ but with a lower priority, its execution will be delayed much further than if $T_{i,j+1}$ were released periodically. As a consequence, the schedulability of the whole system is affected negatively. This is what we call the *clumping effect*.

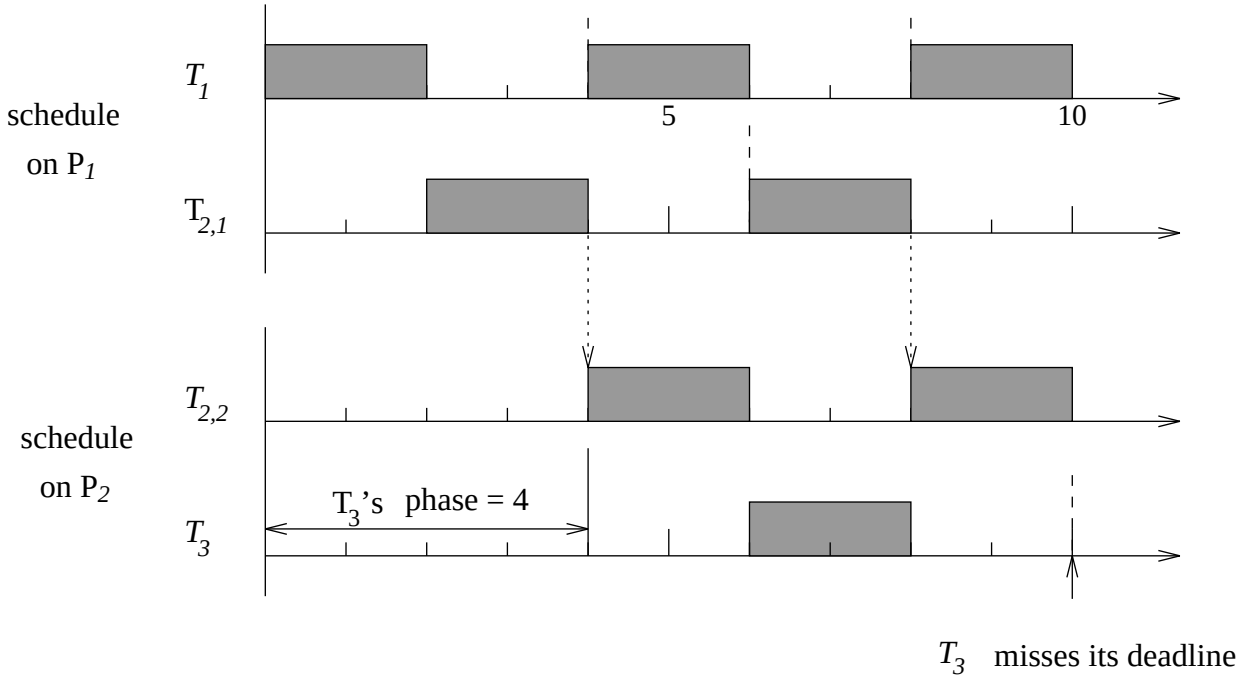

Figure 3: One Schedule of the System in Example 1

When we bound the end-to-end response time of a task, a natural approach is to bound the response times of all its subtasks and then sum the bounds together. The clumping effect, however, causes many difficulties for this approach. We note, from the previous example, that a subtask whose execution is synchronized according to the DS protocol interferes with the execution of a lower priority subtask on the same processor much more than a periodic subtask. To bound the extra interference precisely, we will end up with a mutual dependent system to solve [2]. Furthermore, since many instances of the same subtask can be released almost at the same time, they may interfere with each other (because later instances need to wait until previous instances complete before they can start). Under this situation the bound on the response time of a subtask may be quite pessimistic.

In this report, we take a different approach. For each subtask $T_{i,j}$, we define its *intermediate end-to-end response time*, or its IEER time, to be the length of time from the release of an instance

of $T_{i,1}$ till the completion of the corresponding instance of $T_{i,j}$. We bound the IEER times of subtasks instead of the response times of subtasks, and naturally the bounds on the IEER times of the last subtasks are the bounds on the EER times of the corresponding parent tasks.

3 Algorithm EERT/DS

The algorithm that uses the above described strategy is called Algorithm EERT/DS. Figure 4 shows its structure. The input to Algorithm EERT/DS is the parameters of a set of end-to-end periodic tasks, $\mathbf{T}$, and the output is a set $\mathbf{R}$ of correct upper bounds on the task EER times. According to Algorithm EERT/DS, we first obtain an initial upper bound on the IEER time for each subtask, $\{R_{i,j}\}$. This initial bound may be too optimistic, i.e., the bound may be incorrect because it can be smaller than the actual worst-case IEER time of the subtask. We then feed both the parameters of the tasks and the initial bounds to Algorithm IEERT, which computes a set $\{R'_{i,j}\}$ of new upper bounds on subtask IEER times. If all the new bounds are equal to their corresponding bounds at input, the bound R_{i,n_i} on the IEER time of the last subtask of each task is the (correct) upper bound R_i on the task EER time obtained by Algorithm EERT/DS. Otherwise, we set $R_{i,j}$ to be equal to $R'_{i,j}$ for every subtask, use the new input and apply Algorithm IEERT again. We repeat this iterative process until all the new bounds are equal to their corresponding input bounds. In the rest of this section we first describe Algorithm IEERT, and then prove that the bounds obtained by Algorithm EERT/DS are correct upper bounds on task IEER times when the iteration terminates.

3.1 Algorithm IEERT

We now focus on Algorithm IEERT, which takes as input the parameters of a set of end-to-end periodic tasks in the system and upper bounds on the IEER times of subtasks. The algorithm computes a set of new upper bounds on the IEER times of subtasks as the output. In the following discussion, we focus on subtask $T_{i,j}$, the *target subtask* whose IEER time is to be bounded.

The key technique used in Algorithm IEERT is an extension of *busy period analysis* [4, 5]. We first define a *ϕ-level idle point* and a *ϕ-level busy period.*

Definition 1 *In the schedule of a processor P, a time instant t is a* ϕ-level idle point *if only if every*

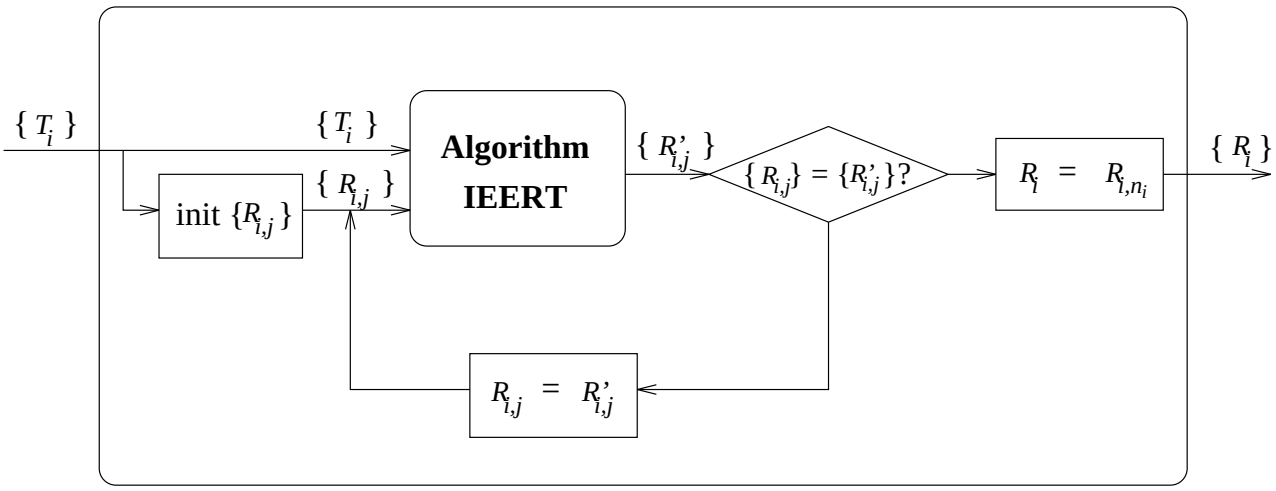

Figure 4: The Diagram of Algorithm EERT/DS

subtask instance that is released on P before time t and has priority higher than or equal to ϕ has completed by time t.

Definition 2 *A* ϕ-level busy period *is a time interval of non-zero length between two consecutive ϕ-level idle points in a schedule.*

Intuitively, a ϕ-level busy period is simply a time interval during which only subtasks with priorities higher than or equal to ϕ execute. However, the intuitive understanding of a busy period can be misleading. For example, in Figure 5, subtask $T_{2,1}$ has higher priority than $T_{1,1}$. In the interval (t_1, t_3) subtasks with priorities higher than or equal to $\phi_{1,1}$ execute continuously. However, (t_1, t_3) is not a single $\phi_{1,1}$-level busy period. Instead, according to the definition, it consists of two $\phi_{1,1}$-level busy periods, (t_1, t_2) and (t_2, t_3), because t_2 is an idle point. In our subsequent discussion, by a busy period, we mean a $\phi_{i,j}$-level busy period in the schedule of the processor where $T_{i,j}$ executes unless otherwise stated. ($T_{i,j}$ is the target subtask.)

We now derive a bound on the maximum duration D of a $\phi_{i,j}$-level busy period and the maximal number M of instances of $T_{i,j}$ that can be in a $\phi_{i,j}$-level busy period. Let $H_{i,j}$ denote the set of subtasks, excluding $T_{i,j}$, that are on the same processor as $T_{i,j}$ and have priorities higher than or equal to $T_{i,j}$. Suppose the maximum duration of a $\phi_{i,j}$-level busy period is D. At the end of the busy period, the time supply since the beginning of the busy period must be equal to the total time

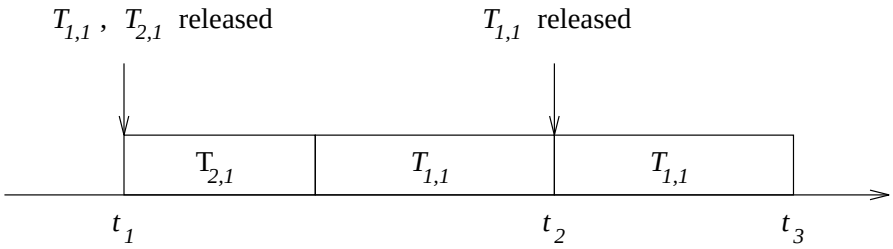

Figure 5: Tricky Situation of Busy Periods in a Schedule

demand of all the subtask instances released and completed in the busy period, i.e.,

$$D = \sum_{T_{u,v} \in H_{i,j} \cup \{T_{i,j}\}} M_{u,v} \times \tau_{u,v} \tag{1}$$

where $M_{u,v}$ is number of instances of $T_{u,v}$ in $(H_{i,j} \cup \{T_{i,j}\})$ that are executed in the busy period.

To bound $M_{u,v}$, we examine Figure 6 which illustrates the releases of instances of $T_{u,v}$ in a busy period, as well as the corresponding instances of the predecessors of $T_{u,v}$. Let $T_{u,v}(1)$ $(T_{u,v}(M_{u,v}))$ denote the first (last) instance of $T_{u,v}$ executed in the busy period. $T_{u,1}(1)$ $(T_{u,1}(M_{u,v}))$ denotes the instance of $T_{u,1}$ corresponding to $T_{u,v}(1)$ $(T_{u,v}(M_{u,v}))$. In the figure, $r_{x,y}(z)$ indicates the release time of subtask instance $T_{x,y}(z)$. Obviously the number of instances of $T_{u,v}$ in the busy period is equal to the number of instances of $T_{u,1}$ released in interval $[r_{u,1}(1), r_{u,1}(M_{u,v})]$. Since subtask $T_{u,1}$ is a periodic subtask, we can bound the number of instances of $T_{u,1}$ released in interval $[r_{u,1}(1), r_{u,1}(M_{u,v})]$ if we can bound the duration of the interval. From the figure, we can derive a upper bound on the duration of interval $[r_{k,1}(1), r_{u,1}(M_{u,v})]$ as follows.

$$r_{u,1}(M_{u,v}) - r_{u,1}(1) < ((t_0 + D) - t_0) + (r_{u,v}(1) - r_{u,1}(1)) \leq D + R_{u,v-1}$$

where $R_{u,v-1}$ is the input upper bound on the IEER time of $T_{u,v-1}$. (By assumption, $R_{i,0}$ is equal to 0 for any i.) Consequently, the number $M_{u,v}$ of instances of $T_{u,v}$ in the $\phi_{i,j}$-level busy period can be bounded by $\lceil (D + R_{u,v-1})/p_u \rceil$. Eq.(1) can thus be re-written as

$$D = \sum_{T_{u,v} \in H_{i,j} + \{T_{i,j}\}} \left\lceil \frac{D + R_{u,v-1}}{p_u} \right\rceil \tau_{u,v} \tag{2}$$

Equation (2) has a solution for D if $\sum_{T_{u,v} \in H_{i,j} \cup \{T_{i,j}\}} \tau_{u,v}/p_u$ is less than 1. A simple iterative process can be applied to obtain the solution. We first rewrite the right-hand side of Eq.(2) as a function of D, denoted by $W(D)$. Let $S_0 = W(0+)$. (Here 0+ stands for an arbitrarily small

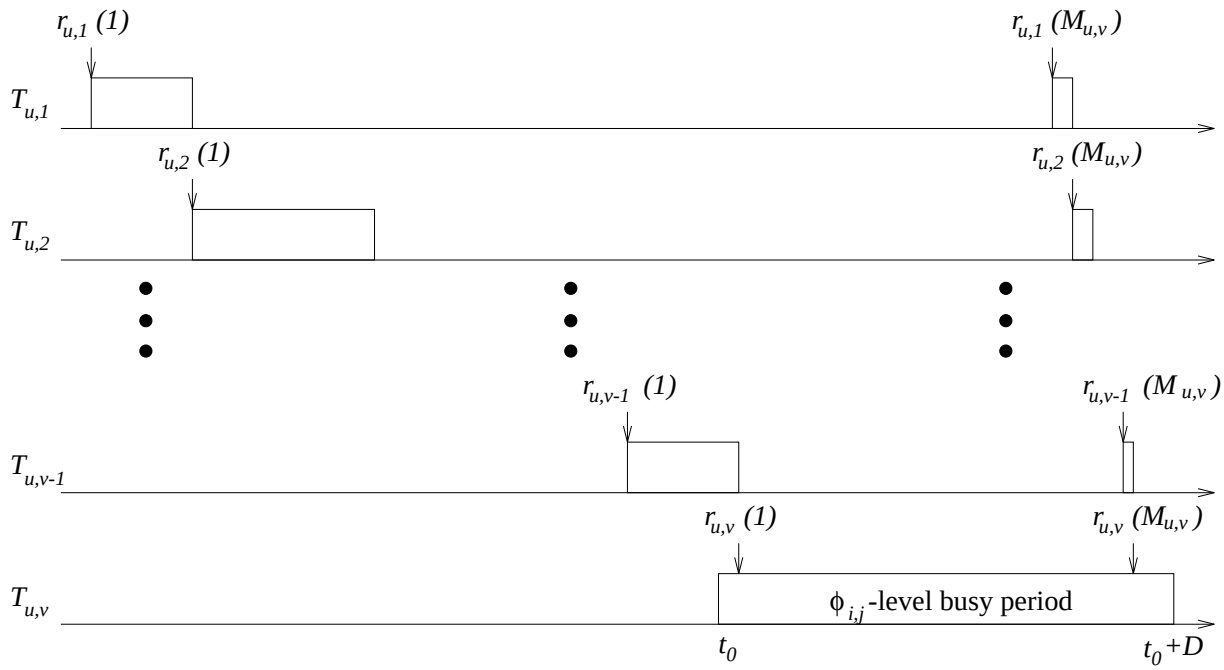

Figure 6: Maximal Duration of the $\phi_{i,j}$-level Busy Period

positive number.) Then for $i = 1, 2, \ldots$, let $S_i = W(S_{i-1})$. If there is a positive solution for D, S_i will eventually converge to the solution. This solution gives the maximum duration of any $\phi_{i,j}$-level busy period in the schedule of the processor where $T_{i,j}$ executes.

We now try to obtain an upper bound on the IEER time for each instance of $T_{i,j}$ in the busy period. Given the maximum duration D of any $\phi_{i,j}$-level busy period, the maximal number M of instances of $T_{i,j}$ in the busy period is given by

$$M = \left\lceil \frac{D + R_{i,j-1}}{p_i} \right\rceil \tag{3}$$

Let $t_0 + C(m)$ $(1 \leq m \leq M)$ be the worst-case completion time of $T_{i,j}(m)$, the mth instance of $T_{i,j}$ in an $\phi_{i,j}$-level busy period. By an analysis similar to the one described in the previous paragraph, the following time demand and supply equation can be established for $C(m)$.

$$C(m) = m\tau_{i,j} + \sum_{T_{u,v} \in H_{i,j}} \left\lceil \frac{C(m) + R_{u,v-1}}{p_u} \right\rceil \tau_{u,v} \tag{4}$$

Again, this equation has a solution for $C(m)$ if $\sum_{T_{u,v} \in H_{i,j}} \tau_{u,v}/p_u$ is less than 1. The iterative process described earlier can be applied to obtain the solution for $C(m)$. From Figure 6, a lower bound on the release time of $T_{i,1}(m)$ is $t_0 - R_{i,j-1} + (m-1)p_i$. Let $R(m)$ denote an upper bound on the IEER

time of $T_{i,j}(m)$, and it is thus computed by

$$R(m) = (t_0 + C(m)) - (t_0 - R_{i,j-1} + (m-1)p_i \quad (5)$$

$$= C(m) + R_{i,j-1} - (m-1)p_i \quad (6)$$

Since $R(m)$ is an upper bound on the IEER time of the mth instance of $T_{i,j}$ in any $\phi_{i,j}$-level busy period, the maximum of $R(m)$ for $m = 1, 2, \ldots, M$ must be an upper bound on the IEER time of $T_{i,j}$ in general. This is the new bound $R'_{i,j}$ computed by Algorithm IEERT. The pseudo-code of the Algorithm IEERT is listed in Figure 7. From the above analysis, we can see that the correctness of $R'_{i,j}$ depends on the correctness of the input bounds. As long as all the input bounds are correct, the output bounds computed by Algorithm IEERT are correct.

Algorithm IEERT

Input :

1. A set $\{T_i\}$ of end-to-end periodic tasks.
2. A set $\{R_{i,j}\}$ of bounds on the IEER times of subtasks.

Output : A set $\{R'_{i,j}\}$ of new bounds on the IEER times of subtasks.

Algorithm :

For each subtask $T_{i,j}$

1. Compute the maximal duration D of a $\phi_{i,j}$-level busy period by solving the following equation

$$D = \sum_{T_{u,v} \in H_{i,j} \cup \{T_{i,j}\}} \left\lceil \frac{D + R_{u,v-1}}{p_u} \right\rceil \tau_{u,v}$$

2. Compute the maximal number M of instances of $T_{i,j}$ in a $\phi_{i,j}$-level busy period by

$$M = \left\lceil \frac{D + R_{i,j-1}}{p_i} \right\rceil$$

3. For $m = 1$ to M do
   (a) Compute $C(m)$ by solving the following equation.

$$C(m) = m\tau_{i,j} + \sum_{T_{u,v} \in H_{i,j}} \left\lceil \frac{C(m) + R_{u,v-1}}{p_u} \right\rceil \tau_{u,v}$$

   (b) Compute $R(m)$ by

$$R(m) = R_{i,j-1} + C(m) - (m-1)p_i$$

4. Compute the new bound $R'_{i,j}$ by

$$R'_{i,j} = \max\{R(m)\}, \quad for\ 1 \le m \le M$$

Figure 7: Pseudo-Code of the Algorithm IEERT

3.2 Algorithm EERT/DS

Let $\mathbf{R} = \{R_{i,j}\}$, and $IEERT(\mathbf{T}, \mathbf{R})$ denote the set $\mathbf{R}'$ of new upper bounds obtained by Algorithm IEERT, i.e., $\mathbf{R}' = IEERT(\mathbf{T}, \mathbf{R})$. Figure 8 lists the pseudo code of Algorithm EERT/DS. In the initialization step, for each subtask $T_{i,j}$, we use sum of the maximum execution times of $T_{i,j}$ and its predecessors as an initial estimate of the bound on its IEER time. Obviously, this estimation may be over-optimistic. After the iteration terminates, the bound on the IEER time of T_{i,n_i} computed during the last iteration is the bound on the EER time of T_i.

Algorithm EERT/DS

Input : Task set $\mathbf{T}$.

Output : The set $\mathbf{R}$ of upper bounds on the EER times of tasks.

Algorithm :

1. For each subtask $T_{i,j}$,
$$R_{i,j} = \sum_{m=1}^{j} \tau_{i,m}$$
$$R'_{i,j} = 0$$
2. Repeat until ($R_{i,j} = R'_{i,j}$ for every subtask $T_{i,j}$)
   (a) $\mathbf{R}' = \mathbf{R}$.
   (b) $\mathbf{R} = IEERT(\mathbf{T}, \mathbf{R}')$.
3. For each task T_i, $R_i = R_{i,n_i}$.

Figure 8: Pseudo-Code of Algorithm EERT/DS

Theorem 1 states an important attribute of Algorithm IEERT, which can be applied directly to prove that Algorithm EERT/DS is correct. The proof of this theorem makes use of the following lemma.

Lemma 1 *Suppose that a subtask instance $T_{i,j}(g)$ completes at time t. If the IEER time of every subtask instance $T_{u,v}(w)$ that completes before t is no greater than $X_{u,v} > 0$, then the IEER time of $T_{i,j}(g)$ is no greater than $X'_{i,j}$, where $\mathbf{X}' = IEERT(\mathbf{T}, \mathbf{X})$.*

Proof:

The correctness of Lemma 1 follows from the extended busy period analysis presented in the previous section. □

Theorem 1 *Suppose for each subtask $T_{i,j}$ there exists some $X_{i,j} > 0$. Let $\mathbf{X} = \{X_{i,j}\}$. If $\mathbf{X} = IEERT(\mathbf{T}, \mathbf{X})$, then $X_{i,j}$ is a correct upper bound on the intermediate end-to-end response time of $T_{i,j}$.*

Proof:

Suppose that the system starts to run at time origin. As an induction basis, we show that the IEER time of the first completed subtask instance in the system is no longer than $X_{i,1}$. We then prove that the IEER time of any subtask instance ($T_{i,j}(g)$) that completes at time t is no longer than $X_{i,j}$ either, provided that the IEER time of every subtask instance $T_{u,v}(w)$ that completes before time t is no longer than the corresponding $X_{u,v}$ value. By induction, we can conclude the theorem is correct.

Induction basis : Starting from the time origin, the first completed subtask instance in the system must be the first instance of the first subtask of a task. Let this subtask instance denoted by $T_{i,1}(1)$. Suppose that $T_{i,1}(1)$ completes at time t. Obviously on the processor where $T_{i,1}$ executes, $T_{i,1}(1)$ has the highest priority among all subtask instances that are released before time t. The response time of $T_{i,1}(1)$ is then equal to its execution time which is less than or equal to $\tau_{i,1}$. On the other hand, by the condition of the theorem that $\mathbf{X} = IEERT(\mathbf{T}, \mathbf{X})$, we can verify that $X_{i,j}$ must be greater than or equal to $\tau_{i,j}$ for every subtask $T_{i,j}$. Therefore, the response time of $T_{i,1}(1)$, or the IEER time of $T_{i,1}(1)$ in this case, must be less than or equal to $X_{i,1}$.

Induction : Suppose an arbitrary subtask instance $T_{i,j}(g)$, which is not the first completed subtask instance in the system, completes at time t, and the IEER time of every subtask instance $T_{u,v}(w)$ that completes before time t is less than or equal to $X_{u,v}$. According to Lemma 1, the IEER time of $T_{i,j}(g)$ is no greater than $X'_{i,j}$, where $\mathbf{X}' = IEERT(\mathbf{T}, \mathbf{X})$. By the condition specified in the theorem, we have $\mathbf{X}' = \mathbf{X}$. Thus the IEER time of $T_{i,j}(g)$ is no greater than $X_{i,j}$. By induction, the IEER time of every subtask instance in this schedule is no greater than $X_{i,j}$. □

Corollary 1 *When it terminates, Algorithm EERT/DS yields correct upper bounds on task EER times.*

3.3 Termination of Algorithm EERT/DS

Unfortunately, Algorithm EERT/DS does not always terminate, as shown by an example in Figure 9. In this example, there are 6 processors and 2 tasks. The period of each task is 3 time units, and the maximum execution time of each subtask is 1 time unit. On each processor the subtask whose corresponding box is drawn higher in the processor circle has a higher priority.

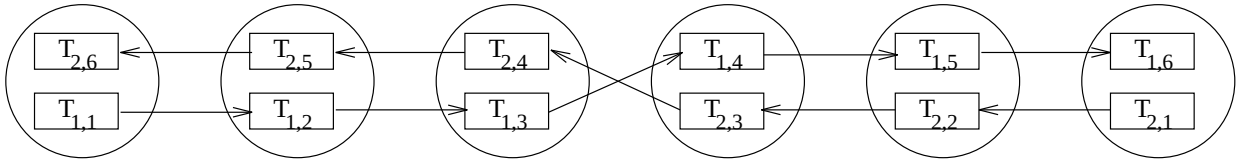

Figure 9: An Example to Show that Algorithm EERT/DS may Never Terminate

We now show that if we apply Algorithm EERT/DS to this system, the algorithm will never terminates. To do so, we follow the iteration steps in Algorithm EERT/DS. Specifically, we start with $T_{1,1}$, $T_{1,2}$ and $T_{1,3}$.

1. Initially, $R_{1,1}$, $R_{1,2}$ and $R_{1,3}$ are equal to 1, 2, and 3 respectively.

2. After the first iteration, $R_{1,1}$, $R_{1,2}$ and $R_{1,3}$ will increase by at least 1 time unit due to the presence of $T_{2,4}$, $T_{2,5}$ and $T_{2,6}$.

3. After at most 5 iterations, $R_{1,3}$, $R_{1,4}$ and $R_{1,5}$ will increase by at least 3 time units compared with their initial values. This is because the increases of $R_{1,1}$, $R_{1,2}$ and $R_{1,3}$ in Step 2 will be propagated to the bounds of later subtasks in the subtask chain. (See Step 3(b) in Algorithm IEERT.)

4. Due to the fact that $R_{1,3}$, $R_{1,4}$ and $R_{1,5}$ have values 3 units larger than their respective initial values, $R_{2,1}$, $R_{2,2}$ and $R_{2,3}$ will increase by at least 1 time unit compared with their initial value in the next iteration step. (See Step 3(a) in Algorithm IEERT.)

5. Similar to Step 3, we will see, at most 5 iterations after step 4, $R_{2,3}$, $R_{2,4}$, and $R_{2,5}$ will increase by at least 3 time units compared with their initial values, due to the propagation of increases of $R_{2,1}$, $R_{2,2}$ and $R_{2,3}$. In the next iteration, $R_{1,1}$, $R_{1,2}$ and $R_{1,3}$ will be further increased by at least 1 time unit. This essentially put us back to the similar situation when we started from step 2.

From the above description, we can see that the iteration in Algorithm EERT/DS essentially forms a positive feedback loop that causes the bounds on subtask IEER times to never converge. In practice, we are never interested in a bound which can be arbitrarily large. In particular, when tasks have relative deadlines, we are not interested in the bounds anymore once we know that the bounds are larger than the deadlines. In this case, we can set up an additional termination condition for Algorithm EERT/DS so that it terminates either when an solution is obtained or when the bounds exceed the relative deadlines.

4 Relation with Previous Work

As we have mentioned earlier, Tindell, and et al. proposed a solution in [2] for a very similar problem. In their approach, the irregular release times are called *release jitters*, and the extra interference to a target subtask caused by release jitters is counted in their extended busy period analysis method [6, 7]. The release jitters are propagated along the subtask chains. The response time of each subtask is thus bounded and the bound on the EER time of each task is simply the sum of the bounds on the response times of all its subtasks.

However, in their approach, although the release jitters of the interfering subtasks, i.e., subtasks that are in $H_{i,j}$, are correctly taken into account, release jitters of the target subtask is not taken into account. Tindell, et al. assumed that the target subtask is periodic [7]. This is only true for the first subtask in each end-to-end task. When a target subtask ($T_{i,j}$) is not the first subtask, itself has release jitters. In an extreme case, several instances of $T_{i,j}$ may be released closely together in time, and the worst-case response time of $T_{i,j}$ is longer than the one when $T_{i,j}$ is periodic, which is obtained by the method in [7]. Since the approach proposed in [2] uses the method in [7] to obtain

an upper bound on the response time of a subtask, we expect that in general Tindell's approach will not yield correct upper bounds on the end-to-end response times of tasks.

5 Summary

In this report, we described an algorithm, Algorithm EERT/DS, that computes upper bounds on the task EER times in an end-to-end system where the DS synchronization protocol and fixed priority scheduling algorithms are used. The DS protocol has many merits: it is simple to implement, it has low run-time overhead, and it yields short average EER times of tasks [1]. Algorithm EERT/DS enables us to analyze the schedulability of a system that uses the DS protocol and, consequently, makes the DS protocol applicable to hard real-time systems.

In this report, we assume that fixed priority scheduling is used. It is possible a dynamic priority scheme might work better in such a systems. In that case, Algorithm EERT/DS is not applicable anymore, and new analysis methods need to be developed in the future.

References

[1] J. Sun and J. W.S. Liu. Synchronization protocols in distributed real-time systems. In *The 16th International Conference on Distributed Computing Systems*, Hong Kong, May 1996.

[2] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 50(2):117–134, April 1994.

[3] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[4] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[5] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[6] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[7] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–152, March 1994.