A New Task Model for Streaming Applications and its Schedulability Analysis

Samarjit Chakraborty
Departement of Computer Science
National University of Singapore
samarjit@comp.nus.edu.sg

Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH) Zürich
thiele@tik.ee.ethz.ch

Abstract

In this paper we introduce a new task model that is specifically targeted towards representing stream processing applications. Examples of such applications are those involved in network packet processing (such as a software-based router) and multimedia processing (such as an MPEG decoder application). Our task model is made up of two parts: (i) a new task structure *to accurately model the software structures of stream processing applications such as conditional branches and different end-to-end deadlines for different types of input data items, and (ii) a new* event model *to represent the arrival pattern of the data items to be processed, which triggers the task structure. This event model is more expressive than classical models such as purely periodic, periodic with jitter or sporadic event models. We then present algorithms for the schedulability analysis of this task model. The basic scheme underlying our algorithms is a generalization of the techniques used for the schedulability analysis of the recently proposed generalized multiframe and the recurring real-time task models.*

1 Introduction

Of late, there has been a lot of interest in the design of hardware and software architectures of embedded systems specifically targeted towards running stream processing applications. Such systems range from hand-held computers, portable audio/video players and mobile phones, to network processors used for implementing complex packet processing tasks in a high-speed router. Many of these devices have very stringent constraints pertaining to cost, size and power consumption, and have posed several challenges towards developing appropriate models, methodologies, languages and tools for designing them (for example, see [5, 6]).

A natural representation of the software structure of a stream processing application is a directed acyclic graph whose nodes represent different processing tasks and the edges represent precedence constraints among these tasks. An example of such a graph, representing a simple network packet processing application is shown in Figure 1. This graph represents the processing of two different classes of packets: VoIP packets, which have real-time constraints

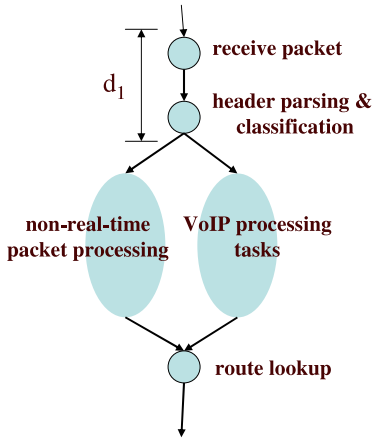

Figure 1: Task graph for a network packet processing application, which processes two different packet types.

on their processing time, and all other packet types, which do not have real-time constraints on their processing time. The processing of any VoIP packet follows the right hand side path in this task graph, and the processing of all other packet types follows the left hand side path in this graph. The arrival of any packet causes an interrupt, which is processed by the *receive packet* task. This is followed by packet header parsing and classification, after which the packet type is known and based on this type either the right or the left hand side of the task graph is executed. Let us assume that all the tasks in this task graph are executed on the same processor. Since the packet type is not known before the first two tasks are completed, there is a deadline (d_1 in the figure) within which the first two tasks are to be executed after the arrival of any packet. d_1 would usually be determined by the line speed or the minimum interarrival time between two packets. VoIP packets have an additional end-to-end deadline, say d_2 (not shown in the figure), within which *all* the relevant tasks in the task graph (starting from *receive packet* to *route lookup*) have to be completed from the arrival time of the packet. d_2 is determined independently from d_1, and depends only on the QoS requirements of the application which is fed by the VoIP packets processed by this task graph.

1530-1591/05 $20.00 © 2005 IEEE

The arrival process of packets which triggers this task graph is usually bursty in nature and can not be accurately captured by event models (such as periodic or sporadic) traditionally studied in the real-time systems literature. Such an arrival process is usually characterized by bounds on the burstiness over different time scales and by long-term arrival rates. In the communication networks domain, such a characterization is done using the theory of *Network Calculus* [3, 4], and we formally define this in Section 2.

Typically, a router will have several network interfaces and each such interface will be associated with a different packet arrival rate. Corresponding to each interface there will be a task graph such as the one shown in Figure 1, which will be triggered by a potentially infinite sequence of packets coming in through this interface. The structure of such a task graph and the different tasks in it might vary from one interface to the other. In the case of a software-based router, all the tasks belonging to such task graphs might be executed on a single processor and it is relevant to ask if all the associated deadlines can be met, given some bounds on the arrival process of packets at each of the interfaces.

Our results and relation to previous work: To the best of our knowledge, none of the task models studied in the real-time systems literature naturally model the setup described above. Most of well known task models either do not model conditional branches, and those that model conditional branches (such as the recently proposed recurring real-time task model [1]) associate deadlines with the individual nodes of the task graphs. Note that the setup described above requires deadlines to be associated with different *paths* in a task graph, all of which start from the *source node*. However, a path can not be "collapsed" into a node, because there might be deadlines associated with subpaths and multiple paths might overlap. Additionally, the event model which triggers our task graphs, has not be studied in the real-time systems literature from the standpoint of schedulability analysis. In fact, the main complication of the schedulability analysis problem that we solve in this paper stems from this event model.

We present algorithms for both static and dynamic priority schedulability analysis. Our basic technique is a generalization of the schedulability analysis developed for the generalized multiframe [2] and the recently proposed recurring real-time task model [1], and is based on the concept of *demand bound* and *resource bound* functions. One of the main contributions of this paper is a method for computing these functions when the task graphs are triggered by our new event model described in the next section. In [2] and [1] these functions were computed for the case where the task graphs are triggered by a sporadic event model.

In the next section we formally define our task model and also describe the concepts of demand bound and resource bound functions. This is followed by our algorithms for schedulability analysis in Section 3. Finally, in Section 4 we outline some directions for future work.

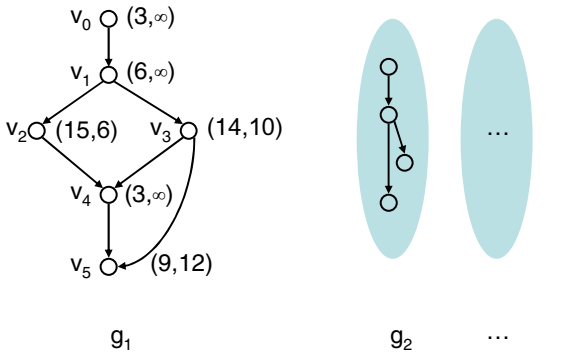

Figure 2: Only one sample task graph $g_1 \in G$ is shown in detail. Each node v_i of g_1 is associated with a tuple $(c(v_i), d(v_i))$, where $c(v_i)$ is the node's worst case resource demand and $d(v_i)$ is its relative deadline.

2 Stream Based Task Model

In this section we formally define the task model we motivated above. For the purpose of schedulability analysis, we consider a set of task graphs G, each of which are triggered independently of each other by data or event streams (where the arrival of a data item may be considered as an *event*). The goal of the schedulability analysis is to ascertain if for all graphs $g \in G$, all the associated deadlines are met for all possible legal triggering sequences. In this paper we are only concerned with schedulability analysis under preemption. An example task set is shown in Figure 2.

Definition 2.1 (Task Graph) *An acyclic, directed task graph $g = (V, A)$ consists of nodes $v \in V$ and edges $a = (v, w) \in A$. A node represents a task of the application, an edge represents a precedence relation. There is a unique source node $v_0 \in V$ having no incoming edges. To each node v there are associated the quantities $c(v) \in \mathbb{R}^{\geq 0}$ and $d(v) \in \mathbb{R}^{>0} \cup \{\infty\}$ which denote the worst case resource demand and the deadline of the task associated to v respectively.*

The interpretation of the precedence relations and deadlines is defined by the dynamic model associated with a task graph. A task graph is instantiated by a timed event, e.g. at time t_0 (which can be the arrival of a packet in the example described in the previous section). The release time of the source task v_0 is t_0. The execution follows a path (sequence of nodes) $(v_0, v_1, \cdots, v_{n-1})$ through g where the nodes are connected by edges $(v_i, v_{i+1}) \in A$. The release time of a task v_{i+1} is the finishing time of task v_i. Each task v requires a maximal resource demand of $c(v)$, given in terms of the number of required processor cycles, the worst case execution time or any other reasonable unit that describes the service of the resource (i.e. the processor on which all

the tasks are executed). In addition, each task has a relative deadline $d(v)$ with respect to the instantiation time t_0 of g. Therefore, its finishing time must be equal or smaller than $t_0 + d(v)$. All events in an event stream need not be processed in the same way. Depending on the type of an event or its associated data, different paths through the task graph may be chosen (see Figure 1). In addition, the processing of each event type might be associated with a different deadline constraint. If the deadline associated with a node is ∞ (see Figure 2), it implies that there are no deadline constraints on the path starting from the source node to this node, although there might be deadlines associated with subpaths of this path.

As we are concerned with the processing of event *streams*, a task graph can be instantiated by successive timed events. We suppose that even a concurrent instantiation is possible, i.e. there can be a new timed event before the last task of the previous instance is completely executed. The timing of any event stream triggering a task graph is bounded by an *arrival function* defined as follows.

Definition 2.2 (Arrival Function) *The arrival function* $\alpha(\Delta) \in \mathbb{R}^{\geq 0}, \Delta \in \mathbb{R}$ *provides an upper bound on the number of events that can arrive within* any *time interval of length* Δ. *In particular, there are at most* $\alpha(\Delta)$ *events within the time interval* $[t, t+\Delta)$ *for all* $t \geq 0$ *and* $\alpha(\Delta) = 0, \forall \Delta \leq 0$.

In practice, $\alpha(\Delta)$ will be defined only for a finite number of values of Δ, and not for all $\Delta \geq 0$. An example of such a specification is: there can not be more than 20 events within any time interval of duration 2 and no more than 25 events within any time interval of duration 5 time units. We denote such a specification using an *arrival sequence*.

Definition 2.3 (Arrival Sequence) *An arrival sequence is given as:*

$$\tilde{\alpha} = \langle \langle \alpha_1, \Delta_1 \rangle, \langle \alpha_2, \Delta_2 \rangle, \cdots, \langle \alpha_n, \Delta_n \rangle \rangle \qquad (1)$$

It is called proper, if in addition $\delta_i < \delta_{i+1}$ *and* $\alpha_i < \alpha_{i+1}$ *for all* $1 \leq i \leq n-1$ *hold.*

An arrival function α *corresponding to an arrival sequence* $\tilde{\alpha}$ *can be determined by the operator* $F_{\tilde{\alpha}}(\Delta)$ *defined as follows*

$$F_{\tilde{\alpha}}(\Delta) = \min\{\alpha_i | \Delta_i \geq \Delta\} \quad \forall\, 0 < \Delta \leq \Delta_n \qquad (2)$$

and $F_{\tilde{\alpha}}(\Delta) = 0, \forall \Delta \leq 0$ *and* $F_{\tilde{\alpha}}(\Delta) = \infty, \forall \Delta > \Delta_n$.

Given an arrival sequence, we can extend and tighten it using the sub-additivity property of arrival functions, i.e. $\alpha(s+t) \leq \alpha(s) + \alpha(t)$. To this end, we at first add linear combinations of tuples to the sequence, i.e. tuples of the form $\langle a, d \rangle$ where $a = \sum_{i=1}^{n} \alpha_i x_i$ and $d = \sum_{i=1}^{n} \Delta_i x_i$ for all $x_i \in \mathbb{Z}^{\geq 0}$. Finally, we remove redundant tuples, i.e. we remove a tuple $\langle a, d \rangle$, if there exists another tuple $\langle \alpha_i, \Delta_i \rangle$ with $\Delta_i \geq d$ and $\alpha_i \leq a$.

In the example arrival sequence mentioned above, we at first have the following sequence: $\langle \langle 20, 2 \rangle, \langle 25, 5 \rangle \rangle$. This sequence can be extended to $\langle \langle 20, 2 \rangle, \langle 40, 4 \rangle, \langle 25, 5 \rangle, \langle 60, 6 \rangle, \langle 45, 7 \rangle, \langle 80, 8 \rangle, \langle 65, 9 \rangle, .. \rangle$. After removing all the redundant tuples, we get $\langle \langle 20, 2 \rangle, \langle 25, 5 \rangle, \langle 45, 7 \rangle, \langle 50, 10 \rangle, \langle 70, 12 \rangle, \langle 75, 15 \rangle, .. \rangle$. Following this concept, two possible representations of example event streams (one discrete and the other continuous) are depicted in Figure 3.

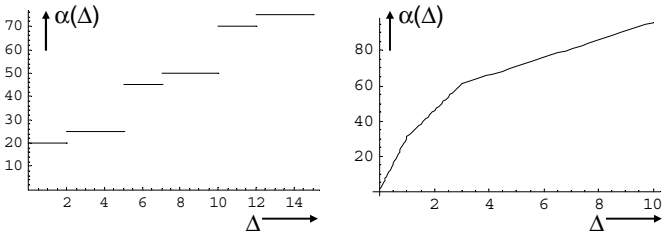

Figure 3: Two possible representations of an event stream associated with a task graph. The right hand side shows a continuous arrival function defined using piecewise linear segments, with three pieces starting at $\Delta = 0$, 1 and 3 with slopes 30, 15 and 5 respectively. The left hand side depicts $F_{\tilde{\alpha}}(\Delta)$ corresponding to the sequence $\tilde{\alpha} = \langle \langle 20, 2 \rangle, \langle 25, 5 \rangle \rangle$ after its sub-additive extension.

In case of several task graphs, we assume that each one $g_k \in G$ is triggered by its own event stream characterized by the corresponding arrival function $\alpha_k(\Delta)$. For example, if the task graph g_1 in Figure 2 is triggered by an event stream characterized by one of the arrival functions in Figure 3, then the additional task graphs (g_2, g_3, ...) in the task set will be triggered by other streams which are characterized by other arrival functions. As mentioned in the previous section, the different event streams and task graphs are independent of each other; they only share the common resource.

For our schedulability analysis, we use the concept of *demand* and *resource bound functions*, similar to the ones introduced by Baruah in [1]. The demand bound function of a task graph describes the minimal amount of resource necessary within any time interval, to satisfy its real-time constraints in the case of a *single instantiation* of the graph. We later introduce the concept of a *composite demand bound function*, which takes into account the triggering of the graph by a *stream* of events.

Definition 2.4 (Demand Bound Function (DBF)) *The demand bound function* $dbf(\Delta) \in \mathbb{R}^{\geq 0}$ *with* $\Delta \in \mathbb{R}^{\geq 0}$ *of a task graph* $g \in G$ *denotes the minimal amount of resource necessary within any time interval of length* Δ *such that all its associated deadlines can be satisfied when the task graph is instantiated by a single event. The composite demand bound function (CDBF)* $dbf^C(\Delta)$ *extends the above definition to the case where the task graph is triggered by a sequence of events whose arrival process is bounded by an arrival function.*

For example, the deadline 6 associated with the node v_2 of the task graph shown in Figure 2 implies that within any time interval of length 6 there must be at least $c(v_0) + c(v_1) + c(v_2) = 24$ resource units available. Otherwise, the relative deadline of v_2 can not be guaranteed. In a similar way, all other deadlines impose additional constraints. However, if the task graph is triggered not once, but by a stream of events, then the resource demand is characterized by a *composite demand bound function* which we formally define in Section 3.2.

Similarly, the resource bound function denotes the maximal load that a task graph can possibly impose on the resource within any time interval of a specified length.

Definition 2.5 (Resource Bound Function (RBF)) *The resource bound function $rbf(\Delta) \in \mathbb{R}^{\geq 0}$ with $\Delta \in \mathbb{R}^{\geq 0}$ of a task graph $g \in G$ denotes the maximal amount of resource load within any time interval of length Δ that can be imposed by a singly instantiated task graph. The composite remand bound function (CRBF) $rbf^C(\Delta)$ extends the above definition to the case where the task graph is triggered by a sequence of events whose arrival process is bounded by an arrival function.*

The use of the above functions in our schedulability analysis problem is shown in Section 3.3.

3 Schedulability Analysis

3.1 Demand and Resource Bound Functions

We will describe the demand and resource bound functions in the form of (proper) sequences similar to the one defined in Def. 2.3, i.e. $\widetilde{dbf} = \langle\langle dbf_1, \delta_1\rangle, \langle dbf_2, \delta_2\rangle, ...\rangle$ and $\widetilde{rbf} = \langle\langle rbf_1, \delta_1\rangle, \langle rbf_2, \delta_2\rangle, ...\rangle$. The values of these functions can be determined using the operator $G_{\widetilde{dbf}}(\Delta)$ defined as follows (the same definition holds for the resource bound sequence also):

$$G_{\widetilde{dbf}}(\Delta) = \max\{dbf_i \mid \delta_i \leq \Delta\} \ \forall\, \Delta \geq \delta_1$$

and $G_{\widetilde{dbf}}(\Delta) = 0$ for all $0 \leq \Delta < \delta_1$.

Now, we can determine a demand bound function of a given task graph g. To this end, let us define $C(v_i)$ as the length of the longest path from the source v_0 of g to v_i using the worst case resource demand $c(v)$ of the node v:

$$C(v_i) = \max\{\sum_{v_j \in P_i} c(v_j) \mid \text{for all paths } P_i\}$$

where P_i denotes a path from source node v_0 to node v_i. Using Def. 2.4, we understand that each node of the task graph g defines a new constraint for the demand bound sequence. Therefore, we at first construct a sequence that contains a tuple for each node v_i of the task graph g, each tuple containing as its elements the length of the longest source path $C(v_i)$ and the deadline $d(v_i)$:

$$\langle\langle C(v_0), d(v_0)\rangle, \langle C(v_1), d(v_2)\rangle, ...\rangle \tag{3}$$

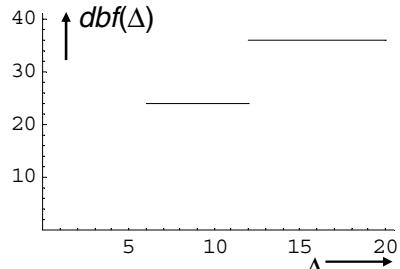

Figure 4: Demand bound function corresponding to the task graph in Figure 2

In order to get a proper demand bound sequence $\widetilde{dbf}$, we must sort these tuples according to the deadlines $d(v_i)$ and remove all tuples that correspond to redundant constraints. In particular, we remove a tuple $\langle c, d\rangle$, if there exists another tuple $\langle c_i, d_i\rangle$ with $d_i \leq d$ and $c_i \geq c$.

In a similar way, we can conclude that the resource bound sequence is simply given by

$$\widetilde{rbf} = \langle\langle C^{max}, 0\rangle\rangle \tag{4}$$

where C^{max} denotes the longest source path in the task graph g, i.e. $C^{max} = \max\{C(v_i) | v_i \in V\}$.

Using the task graph g_1 shown in Figure 2, we first obtain the sequence (3), which is equal to $\langle\langle 24, 6\rangle, \langle 23, 10\rangle, \langle 36, 12\rangle\rangle$. Here each tuple corresponds to one node with a finite relative deadline, i.e. nodes v_2, v_3 and v_5. As the second tuple corresponds to a redundant constraint, we obtain $\widetilde{dbf} = \langle\langle 24, 6\rangle, \langle 36, 12\rangle\rangle$. The resource bound function is constant, i.e. $rbf(\Delta) = 36$, $\forall \Delta > 0$. The demand bound function corresponding to the task graph g_1 in Figure 2 is shown in Figure 4.

Now, it remains to determine the composite demand and resource bound functions by taking into account the instantiation of a task graph by an event stream that is bounded by an arrival function $\alpha(\Delta)$ or an arrival sequence $\tilde{\alpha}$.

3.2 Composite Demand and Resource Bound Functions

The computation of the composite demand bound function of a task graph is given by the following theorem.

Theorem 3.1 (Composite Demand Bound Function) *Given a demand bound sequence $\widetilde{dbf} = \langle\langle dbf_1, \delta_1\rangle, \langle dbf_2, \delta_2\rangle, ...\rangle$ and an arrival function as defined in Def. 2.2. Then the composite demand bound function according to Def. 2.4 is given by*

$$dbf^C(\Delta) = \sum_{i=1}^{m} (dbf_i - dbf_{i-1}) \cdot \alpha(\Delta - \delta_i)$$

where $dbf_0 = 0$.

Proof: The proof uses induction on the number of tuples in the demand bound sequence. Let us first suppose that $\widetilde{dbf} = \langle\langle dbf_1, \delta_1\rangle\rangle$. Then for one triggering event, the demand is 0 for all $0 \leq \Delta < \delta_1$ and it is dbf_1 for all $\delta_1 \leq \Delta$.

Now, there are at most $\alpha(\Delta)$ events in any interval of size Δ. Each event triggers the graph g and generates a demand according to the demand bound sequence $\widetilde{dbf}$. Further note, that the demand of independent tasks is additive. Using this, one can conclude that the composite demand in an interval of $\Delta+\delta_1$ is at most $dbf_1 \cdot \alpha(\Delta)$, i.e $\alpha^D(\Delta+\delta_1) = dbf_1 \cdot \alpha(\Delta)$. As a result, we obtain that $dbf^C(\Delta) = 0$ for $\Delta < \delta_1$ and $dbf^C(\Delta) = dbf_1 \cdot \alpha(\Delta - \delta_1)$ for $\Delta \geq \delta_1$. Using Eq. (2), we obtain the relation given in the theorem.

Now, we will use induction on the number of tuples in the DBF sequence. Let us suppose that Theorem 3.1 holds for m elements of the sequence $\widetilde{dbf}^m = \langle\langle dbf_1, \delta_1\rangle, \cdots, \langle dbf_m, \delta_m\rangle\rangle$, i.e. $dbf^C_m(\Delta) = \sum_{i=1}^{m}(dbf_i - dbf_{i-1}) \cdot \alpha(\Delta - \delta_i)$. From the transformation between a sequence and the corresponding function in Eq. (2) we know that $dbf^{m+1}(\Delta) - dbf^m(\Delta) = 0$ for $\Delta < \delta_{m+1}$ and $dbf^{m+1}(\Delta) - dbf^m(\Delta) = dbf_{m+1} - dbf_m$ if $\Delta \geq \delta_{m+1}$. Taking into account again the additivity of demands, we can conclude the following. The difference in demand between m and $m+1$ sequence elements is given by the demand bound function that is defined by the sequence $\langle\langle dbf_{m+1} - dbf_m, \delta_{m+1}\rangle\rangle$. Following the arguments in the first paragraph of the proof, this leads to an additional composite demand of $(dbf_{m+1} - dbf_m) \cdot \alpha(\Delta - \delta_{m+1})$ if the task graph g is triggered by an event stream bounded by $\alpha(\Delta)$. Therefore, $dbf^C_{m+1}(\Delta) = dbf^C_m(\Delta) + (dbf_{m+1} - dbf_m) \cdot \alpha(\Delta - \delta_{m+1}) = \sum_{i=1}^{m+1}(dbf_i - dbf_{i-1}) \cdot \alpha(\Delta - \delta_i)$. ■

If the triggering event stream is described by an arrival sequence according to Def. 2.3, then we can construct a sequence representation of the composite demand bound function by including all tuples of the form

$$\langle (dbf_i - dbf_{i-1})\alpha_j, \Delta_j + \delta_i\rangle \quad \forall\, 1 \leq i \leq m,\ 1 \leq j \leq n$$

and then removing all redundant constraints, i.e. we remove a tuple $\langle c, d\rangle$, if there exists another tuple $\langle c_i, d_i\rangle$ with $d_i \leq d$ and $c_i \geq c$.

The composite resource bound function can be constructed in a similar way.

Theorem 3.2 (Composite Resource Bound Function) *Given a resource bound function $\widetilde{rbf} = \langle\langle rbf_1, \delta_1\rangle, \langle rbf_2, \delta_2\rangle, ...\rangle$ and an arrival function in accordance with Def. 2.2. Then the composite resource bound function according to Def. 2.5 is given by*

$$rbf^C(\Delta) = \sum_{i=1}^{m}(rbf_i - rbf_{i-1}) \cdot \alpha(\Delta - \delta_i)$$

where $rbf_0 = 0$.

Using Eq. (4), the composite resource bound function can be determined as $rbf^C(\Delta) = C^{max} \cdot \alpha(\Delta)$, or

$$\widetilde{rbf}^C = \langle\langle C^{max}\alpha_1, \Delta_1\rangle, ..., \langle C^{max}\alpha_m, \Delta_m\rangle\rangle$$

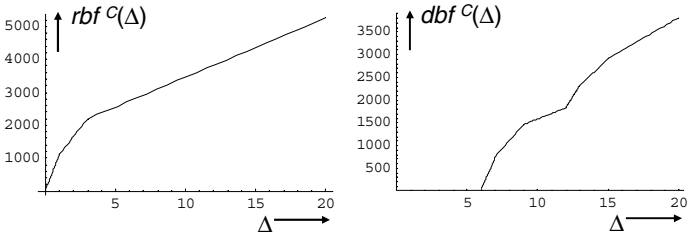

Figure 5: Composite resource and demand bound functions corresponding to the task graph g_1 in Figure 2 and the arrival function shown at the right hand side of Figure 3.

in case of an arrival sequence in accordance with Def. 2.3. As an example, let us suppose that the task graph g_1 in Figure 2 is triggered by an event stream characterized by the continuous arrival function shown at the right hand side of Figure 3. Based on the demand bound function in Figure 4, Figure 5 shows the resulting composite resource and demand bound functions.

3.3 Conditions for Schedulability

In order to perform a schedulability analysis for our task, we make use of known results based on the resource and demand bound functions for static and dynamic priority schedulability analysis (see [1] for details). We can use these results because of our task independence assumptions.

Let us suppose that the task graphs $g_k \in G$ for $1 \leq k \leq N$ are ordered according to the priority of their tasks, i.e. all nodes of g_k have a higher priority than those of g_{k+1}. Moreover, we assume that the resource (or the processor) delivers at least $\beta(\Delta)$ resource units within any time interval of length Δ. For example, if the resource demand is directly given in terms of time, then we have $\beta(\Delta) = \Delta$ for an unloaded processor. We then have the following result:

A set of task graphs $g \in G$ is schedulable under a fixed priority scheme, if and only if

$$dbf^C_k(\Delta) \leq \sup_{0 \leq \lambda \leq \Delta} \left\{ \beta(\lambda) - \sum_{j=1}^{k-1} rbf^C_j(\lambda) \right\} \tag{5}$$

holds for all graphs g_k and for all $\Delta > 0$. Intuitively, the right hand side represents the remaining resource capability after the maximum resource load due to all higher priority tasks has been taken into account. The left hand side represents the composite demand by the tasks of graph g_k. Using this test, one can also determine a feasible priority assignment efficiently (again, see [1] for details).

In a similar way, we can perform a schedulability analysis for preemptive dynamic priority scheduling. In this case, a set of task graphs $g_k \in G$ is schedulable if and only if

$$\sum_{k=1}^{N} dbf^C_k(\Delta) \leq \beta(\Delta) \tag{6}$$

holds for all $\Delta \geq 0$. The right hand side represents the accumulated resource demand of all tasks graphs g_k and the left

hand side denotes the available resource capacity. In the particular case where the task graphs are scheduled using an EDF scheduling policy, the relative deadlines associated with the nodes of a graph might have to be adjusted (since some of them might be equal to ∞) in a way such that they are still compatible with the demand bound function. The new relative deadline of a node v_i can be assigned as the minimum of its specified deadline $d(v_i)$ and deadlines of all its direct and indirect successors in the task graph. In the case of the graph g_1 shown in Figure 2, the new deadlines are $d(v_0) = 6$, $d(v_1) = 6$, $d(v_2) = 9$, $d(v_3) = 10$, $d(v_4) = 12$ and $d(v_5) = 12$. The nodes of this graph can now be scheduled with these deadlines.

Just for the purpose of illustration, let us assume that there is only one input stream with a composite demand bound function as depicted in Figure 4. Figure 6 represents a schedulability test for EDF according to Eq. (6), with the assumption that the computing resource can deliver 195 resource units per time unit, i.e. $\beta(\Delta) = 195 \cdot \Delta$. As it turns out, the stream is just schedulable.

The schedulability tests given above can also be applied to sequence representations of the composite demand and resource bound functions. But the way they have been formulated so far, the relations in Eqs. (5) and (6) need to be checked for all $\Delta \geq 0$, at least in principle. Even if we restrict ourselves to $\Delta \in \mathbf{Z}^{\geq 0}$, this is virtually infeasible. Therefore, we are interested in a maximal value Δ_{max} with the following property. If the inequalities hold for all $0 \leq \Delta \leq \Delta_{max}$ then they hold for all $\Delta \geq 0$. In what follows, we derive such a bound for dynamic priority schedulability analysis. The derivation for the fixed priority case is similar.

We bound all the arrival functions α_k (associated with the graph $g_k \in G$) by a linear function: $\alpha_k(\Delta) \leq s_k^\alpha + t_k^\alpha \cdot \Delta$ for all $\Delta \geq 0$. In a similar way, we can bound $\beta(\Delta)$ using $\beta(\Delta) \geq s^\beta + t^\beta \cdot \Delta$. Now, we know from Eq. (3) that the demand bound functions can be bounded by $\widetilde{dbf}_k = \langle\langle C_k^{max}, d_k^{min} \rangle\rangle$ where C_k^{max} denotes the maximal accumulated weight of any path in the task graph g_k using weights $c(v_j)$ and d_k^{min} denotes the smallest deadline in the task graph g_k. Therefore, we can derive the bound

$$\sum_{k=1}^{N} dbf_k^C(\Delta) \leq \sum_{k=1}^{N} C_k^{max}(s_k^\alpha - d_k^{min} t_k^\alpha) + (\sum_{k=1}^{N} C_k^{max} t_k^\alpha)\Delta$$

From Eq. (6), we can conclude that the above expression needs to be equal to $s^\beta + t^\beta \cdot \Delta_{max}$, and therefore

$$\Delta_{max} = \frac{(\sum_{k=1}^{N} C_k^{max}(s_k^\alpha - d_k^{min} t_k^\alpha) - s^\beta}{t^\beta - (\sum_{k=1}^{N} C_k^{max} t_k^\alpha)}$$

In our running example, we have $s^\alpha = 46$, $t^\alpha = 5$ (see Figure 3) and $s^\beta = 0$, $t^\beta = 195$ (see Figure 6). From the task graph in Figure 2, we conclude that $C^{max} = 36$ and $d^{min} = 6$. Therefore, we obtain $\Delta_{max} = 36(46-30)/(195-36*5) =$ 38.4, i.e. we would need to check the inequality given by Eq. (6) for all $0 \leq \Delta \leq 38.4$. The same holds, if the bound functions are represented in the form of sequences.

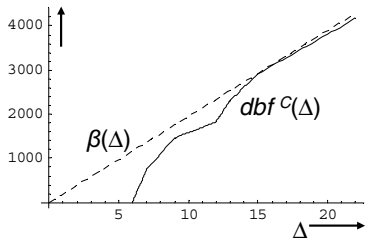

Figure 6: EDF schedulability test for the composite demand bound function given in Figure 4 and $\beta(\Delta) = 195 \cdot \Delta$.

4 Concluding Remarks

In this paper we introduced a new task model that naturally captures some of the essential properties of stream processing applications. For this class of applications, we believe that our model is more pragmatic compared to the task models recently studied in the real-time systems literature (such as [1]), although they can also model program structures like conditional branches. We did not present a formal complexity analysis of our scheduling algorithms. In all realistic setups, there will only be a small number of deadline constraints associated with any task graph (although the graph might have exponential number of paths from its source to its sink node). Bounds on typical event streams will also be specified using a small number of tuples (such as a burst and a long-term rate). In such cases, our schedulability analysis will certainly have a reasonably low computational cost. It should be noted that even in such cases, it might not be possible to ascertain schedulability by avoiding the test we presented, and instead using some straightforward calculation. In future, we plan to extend this model to incorporate variable task execution times (packet payload processing) and timeouts.

References

[1] S. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[2] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[3] R. Cruz. A calculus for network delay, Parts 1 & 2. *IEEE Transactions on Information Theory*, 37(1), 1991.

[4] J.-Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer, 2001.

[5] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004.

[6] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. In *11th Conference on Compiler Construction (CC)*, LNCS 2304, 2002.