End-To-End Deadlines over Dynamic Topologies

Victor Millnert
Lund University, Sweden

Johan Eker
Lund University, Sweden
Ericsson Research, Sweden

Enrico Bini
University of Turin, Italy

Abstract

Despite the creativity of the scientific community and the funding agencies, the underlying model of computation behind IoT, WSN, cloud, edge, fog, and mist is fundamentally the same; Computational nodes which are dynamically interconnected to form a system in where both processing capacity and connectivity may vary over time. On top of such a system, we consider applications that need packets to flow along a path and adhere to end-to-end deadlines. This application model is motivated by both control and automation systems, as well as telecom systems. The challenge is to guarantee end-to-end deadlines when allowing nodes and applications to join or leave.

The mainstream, and to some extent natural, approach to this is to relax the stringency of the constraint (e.g. use probabilistic guarantees, soft deadlines). In this paper we take a different approach and keep the end-to-end deadlines as hard constraints and instead partially limit the freedom of how nodes and applications are allowed to leave and join. We present a theoretical framework for modeling such systems along with proofs that deadlines are always honored.

2012 ACM Subject Classification Computer systems organization → Cloud computing; Networks → Cloud computing

Keywords and phrases Cloud, real-time, end-to-end latency guarantee, end-to-end response time guarantee, dynamic network

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2019.10

1 Introduction

Cloud computing plays a pivotal role in the ongoing digitalization of both the industry and the society at large. This transformation borrows many technologies and concepts from traditional cloud applications (mail services, ride sharing, media streaming, e-commerce etc.). The focus for these applications is commonly availability, scalability, and price/performance. While response time is of great concern for such applications, determinism is usually not. For the next generation of cloud-applications, such as industrial automation, collaborative traffic, and telecom systems, predictable timing is crucial. In order to secure a successful transformation and allow such timing-critical systems, the underlying cloud infrastructure must be able to guarantee predictable end-to-end response-times.

One example of such an application could be a dynamically reconfigurable production cell in a manufacturing plant, where the elements in the production cell are connected to the cloud and controlled centrally. The cloud provides large-scale compute and storage capacity. Cloud back-end systems are typically implemented as a service meshes consisting of networks of interconnected microservices. The production cell may be dynamically configured to adapt to changes, such as hardware failures, or respond to external events, etc. Elements may thus dynamically join or leave a cell. Figure 1 illustrates such a production cell with industrial robots connected to the cloud, which provide services for automation, analytics, artificial intelligence and machine learning algorithms. The topology of the service network may

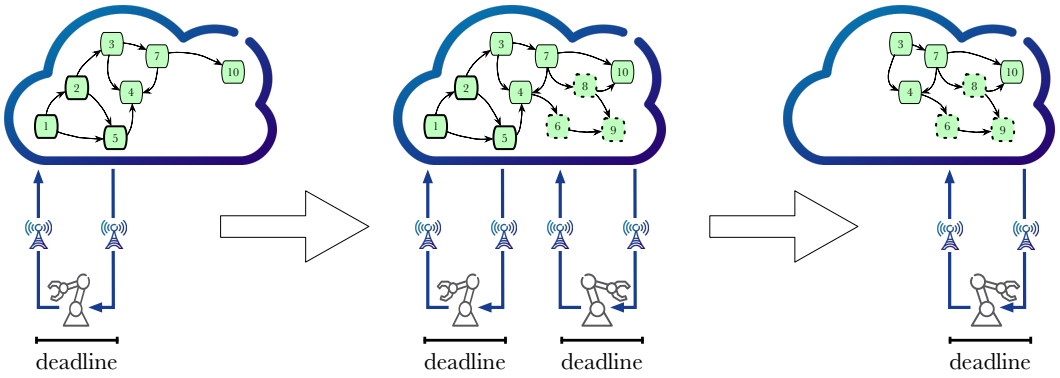


Figure 1 Illustration of configurable, and mobile, manufacturing cells connected to a set of smart services in the cloud. It highlights the changes of the network topology which arises when new robots, and cloud services, join and leave the system. With the upcoming 5G standard it becomes possible to establish a low-latency wireless connection between the robots and the cloud. However, in order to guarantee the required low end-to-end response-time for the robots, there still remains a challenge to ensure a low end-to-end response-time within the cloud, especially during the transitions of the network topology.

therefore change over time as robots join and leave the network. As mentioned earlier, the challenge is to still guarantee that the end-to-end deadline required by the robots are always met, despite dynamic changes of the network topology.

This paper addresses the challenge of allowing network churn and still ensures predictable end-to-end response times within the cloud. A framework that allows for transitions of the network topology is presented. Moreover, it is formally proved that the suggested framework guarantees that it will never violate any timing constraints.

2 The system model

The focus of this paper is to develop a framework where applications are implemented as flows of packets through a network of nodes. Each node offers a service to the incoming packets. The goal is to manage the on-line arrival/departure of flows and the on-line arrival/departure of nodes of the network in a way such that end-to-end deadlines of packets are honored, while allowing for topology changes.

Section 2.1 presents definitions and assumptions with respect to the services offered by *nodes*. Applications are then defined as a set of interconnected services and referred to as *flows*. Definitions and assumptions on flows are given in Section 2.2. Finally, in Section 2.3, we present some assumptions on how these flows and nodes interact with the resource manager.

2.1 Model of the nodes

A node represents an entity that offers a service to the incoming packets. The time taken by a node to provide the service to an incoming packet is given in Def. 1 below. The nodes in the system are denoted by $\mathcal{V}(t) \subset \mathbb{N}$, which is the set of indices of the nodes present at time t. The terms "node" and "vertex" are used interchangeably[1].

[1] We may use the term "node" when we refer to its capacity to process packets, while the term "vertex" is more often used when referring to the topological structure of the network.

▶ **Definition 1** (Response time). *We define the* response time $R_i(t)$*, as the time taken by a packet entering node i at time t to be processed.*

From the definition above, we remark the following facts:

- The response time $R_i(t)$ accounts for all sources of delay possibly occurring within node i (queuing, interference, processing, etc.).
- All incoming packets are treated in the same way regardless of the flow they belong to. Differentiantig packets depending on the application they belong to is feasible. This would require to attach the flow index j to the response time, which would become then $R_{i,j}(t)$. However, this choice poses a notational challenge only with no conceptual added value. For this reason we believe that letting the packet response time depend only on the node (and time) does not significantly impact the applicability of the results. We leave the case of per-flow response time to future works.
- Moreover, in contrast to a large body of the research on real-time systems, this paper does not address how the response time may be computed based on the amount of incoming workload (due to the arrival of packets) and the amount of processing capacity of a node, etc. The interested readers can refer to a vast relate literature addressing this aspects [12, 18, 7, 16, 17, 19, 25, 3].
  Rather, the focus is on the interactions between nodes aimed at guaranteeing end-to-end deadlines in the context of a dynamic network.

Naturally, the service provided to packets by a node might include some minimum requirements, which determines a lower bound to the response time of the service time of a node. The node is unable to process packets in less time than this value. Hence, to model the minimum time needed by a node to process a packet, we introduce the following definition.

▶ **Definition 2** (Response time lower bound). *We define the* response time lower bound $\underline{R}_i$*, as the minimum a packet may take to be processed by node i.*

In practice, $\underline{R}_i$ may represent the pure processing time of a packet, without any interference or delay of any kind.

Finally, we assume that the node is capable of *controlling its response-time*, as stated below in Assumption 1. This assumptions is backed by the vast body of research for different ways of controlling the response-time of cloud services. For instance, in [18] they use the concept of "brownout control" to ensure that the response-time of a server is within a desired limit. In [7] control theory is used to modify the processing capacity of the web servers to control response times. In [17] a combination of scaling the resources of the nodes with an admission control is used to ensure that the deadlines of the nodes in the network are met. More interesting work addressing this is found in [19, 25, 3].

▶ **Assumption 1** (Response-time control). *We assume that a node i is capable of controlling its response-time $R_i(t)$ such that it is always below a deadline $D_i(t) \geq \underline{R}_i$, i.e., such that*

$$\forall t \geq 0,\ \forall i \in \mathcal{V}(t), \quad R_i(t) \leq D_i(t). \tag{1}$$

2.2 Model of the flows

An application in the system is modeled as a *flow* indexed by some $j \in \mathcal{F}(t)$ and characterized by a *path* and an *end-to-end deadline*, properly defined next.

▶ **Definition 3** (Path). *The* path *of a flow $j \in \mathcal{F}(t)$ is defined by the sequence $p_j : \{1, \dots, \ell_j\} \to \mathcal{V}(t)$, with $\ell_j \geq 1$ being the* length of the path*, such that*

$$\forall i = 1, \dots, \ell_j - 1, \quad (p_j(i), p_j(i+1)) \in \mathcal{E}(t), \tag{2}$$

where $\mathcal{E}(t) \in \mathcal{V}(t) \times \mathcal{V}(t)$ are the current edges between the nodes of the network.

By Def. 3, it follows that $p_j(i)$ is the i-th node on the path of flow j. It should be noted that Eq. (2) enforces the existence of an edge of the graph between two consecutive nodes in a path. It should be noted that this paths may traverse a node more than once, which allows us to capture typical client-server sessions. With a slight abuse of notation, we may denote the image of the map p_j, which is the set of nodes touched by path j, by p_j only, rather than $p_j(\{1, \dots, \ell_j\})$.

We remark that the type of application addressed in this paper is borrowed from cloud microservices, where each service provides a unique value to the travelling packet. Hence, the route of packets belonging to an application is known and must not be decided at run time.

▶ **Definition 4** (End-to-end deadline). *We define the* end-to-end deadline $\mathcal{D}_j$ *of the flow $j \in \mathcal{F}(t)$ as the maximum time a packet may take to be processed by all nodes along the path p_j.*

Finally, a flow j is also characterized by an *end-to-end response-time* $\mathcal{R}_j(t)$, which is the time taken by a packet entering the first node $p_j(1)$ of the flow at time t to be processed by all nodes of the path p_j. However, before properly defining $\mathcal{R}_j(t)$, we need to introduce the *mid-path response-time* $\mathcal{R}_{j,i}(t)$, that is the time needed by a packet that entered flow j at time t to pass through the first i nodes of the path p_j. Formally, this quantity is defined recursively by

$$\mathcal{R}_{j,i}(t) = \begin{cases} R_{p_j(1)}(t) & \text{if } i = 1 \\ \mathcal{R}_{j,i-1}(t) + R_{p_j(i)}\big(t + \mathcal{R}_{j,i-1}(t)\big) & \text{otherwise.} \end{cases} \tag{3}$$

The intuition of (3) is quite straightforward: the time it takes a packet to traverse the first i nodes is equal to the time required to traverse the first $i-1$ nodes plus the response-time of the i-th node. The *end-to-end response-time* of a flow j is therefore given by $\mathcal{R}_{j,\ell_j}(t)$, which we compactly denote by $\mathcal{R}_j(t)$.

2.3 Model of the dynamic network

Flows and nodes may join and leave the network at run time. Hence, we define the network as follows.

▶ **Definition 5** (Network). *We define the* network $\mathcal{G}(t)$ *of the system at time t as the set of* nodes*,* directed edges*, and* flows *present at that time: $\mathcal{G}(t) = \{\mathcal{V}(t),\ \mathcal{E}(t),\ \mathcal{F}(t)\}$.*

Since we aim at guaranteeing end-to-end deadlines, the requests to join or leave must be properly handled by a *resource manager*, which manages the network (as illustrated in Figure 2). If not properly handled, the risk is that newly admitted flows may cause overload or the uncontrolled departure of nodes may disconnect the network.

The interactions between the resource manager and the flows are as follows:

- A flow $j \notin \mathcal{F}(t)$ may *request to join* the network. Such an instant is denoted by $f_j^{\mathsf{rq}+}$. When a new flow issues such a request, it also communicates to the resource manager the following information:
  1. its path p_j
  2. its end-to-end deadline $\mathcal{D}_j$.

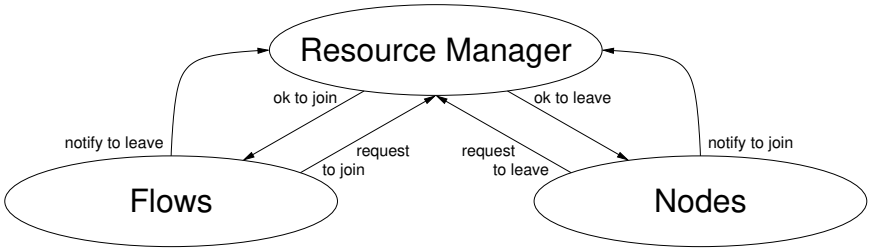


Figure 2 Scheme of interactions between the resource manager (which manages the network), the flows, and the nodes.

- After the request by a flow to join, the resource manager:
  1. accepts the flow j to the network at an instant $f_j^{\mathsf{ok}+}$ (which $\geq f_j^{\mathsf{rq}+}$), if feasible, or
  2. rejects the flow j immediately, if not feasible.

  Details on the admission of new flows based on its characteristics and the current state of the network are given in Section 4.
- A flow $j \in \mathcal{F}(t)$ may *notify and leave* the network at any time that we denote by f_j^-. In fact, it is only advantageous to let a flow (and its constraint) to leave.
- The resource manager may notify a flow $j \in \mathcal{F}(t)$ that it has to leave the network. This might, for instance, happen if a node along the path p_j requests to leave the network. In such a case, the resource manager is no longer able to provide the requested services of the flow j.

The interactions between the resource manager and the nodes (which are the vertices of the graph) are as follows:

- A node $i \notin \mathcal{V}(t)$ may *notify and join* the network at any time. We denote such an instant by v_i^+. Since a node is bringing a new service to the network, there is no admission control to its request to join.
- A node $i \in \mathcal{V}(t)$ may *request to leave* the network to the resource manager. The instant of such a request is denoted by $v_i^{\mathsf{rq}-}$.
- The resource manager lets a node leave only at time $v_i^{\mathsf{ok}-}$ (which is $\geq v_i^{\mathsf{rq}-}$). The time between $v_i^{\mathsf{rq}-}$ and $v_i^{\mathsf{ok}-}$ is needed by the resource manager to allow flows going through node i to properly exit.

Problem formulation

The problem formulation in this paper can now formally be summarized as follows;

- control when/how nodes are allowed to leave the network,
- control when/how flows are allowed to join the network,
- control the node deadlines $\mathbf{D}(t) = [D_i(t)]_{\forall i \in \mathcal{V}(t)}$,

such that the end-to-end deadlines of all the flows in the network are met:

$$\forall t \geq 0,\ \forall j \in \mathcal{F}(t), \quad \mathcal{R}_j(t) \leq \mathcal{D}_j. \tag{4}$$

3 Static networks

In this section, we introduce a protocol which allows for *dynamic deadlines* in static networks. By a static network we mean one where no nodes or flows join or leave the network, that is $\forall t \geq 0,\ \mathcal{G}(t) = \mathcal{G}$. The traffic rates and response times of nodes may change dynamically.

However, the topology is static: the nodes, the flows and their end-to-end deadlines do not change over time. Hence, we can drop the time dependency of the set $\mathcal{E}(t) = \mathcal{E}$ of edge, the set $\mathcal{V}(t) = \mathcal{V}$ of nodes, and the set $\mathcal{F}(t) = \mathcal{F}$ of flows. However, node deadlines may change to accommodate variations in the workload.

To give some intuition of the challenges of guaranteeing end-to-end deadlines in a system with dynamic node deadlines (even for a static network), we present a simple example in Section 3.1. In Section 3.2 we propose a solution, which allows for dynamic node deadlines and still provides guarantees on the end-to-end deadlines. Naturally, this is also proved in Theorem 6. Finally, in Section 3.3 we adapt the opening example of Section 3.1 such that it uses the method suggested in Section 3.2 and show that it is then able to guarantee that all the end-to-end deadlines are met for all times.

3.1 Example – issues with dynamic deadlines

If node deadlines were constant then a static assignment of node deadlines, equal to any vector of node deadlines $\mathbf{D} = \{D_i\}_{\forall i \in \mathcal{V}}$ satisfying

$$\mathbf{D} \in \overline{\mathbb{D}}(\mathcal{G}) = \left\{ D_i \in \mathbb{R}^+ : \forall i \in \mathcal{V},\ D_i \geq \underline{R}_i, \quad \forall j \in \mathcal{F},\ \sum_{i=1}^{\ell_j} D_{p_j(i)} \leq \mathcal{D}_j \right\}. \tag{5}$$

would guarantee no end-to-end deadlines to be missed. The intuition behind Eq. (5) is simple: the sum of the node deadlines along the path p_j of a flow j cannot exceed the end-to-end deadline $\mathcal{D}_j$ of the path j. We remark that the set $\overline{\mathbb{D}}(\mathcal{G})$ is convex, since it is the polytope built from the intersection among linear half-spaces. This also implies that every line between any two points in $\overline{\mathbb{D}}(\mathcal{G})$ belongs to $\overline{\mathbb{D}}(\mathcal{G})$.

Let us now consider the issues of performing a transition from some deadline assignment $\mathbf{D}(t_1) \in \overline{\mathbb{D}}(\mathcal{G})$ to another assignment $\mathbf{D}(t_2) \in \overline{\mathbb{D}}(\mathcal{G})$, hence with both the starting and ending node deadlines belonging to $\overline{\mathbb{D}}(\mathcal{G})$. Since $\overline{\mathbb{D}}(\mathcal{G})$ is convex then the linear transition of node deadlines

$$\mathbf{D}(t) = \mathbf{D}(t_1) \times \frac{t_2 - t}{t_2 - t_1} + \mathbf{D}(t_2) \times \frac{t - t_1}{t_2 - t_1}$$

always belongs to $\overline{\mathbb{D}}(\mathcal{G})$ for all $t \in [t_1, t_2]$.

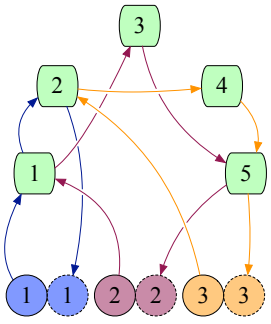


Figure 3 Example of network. Nodes are represented by light yellow boxes. Flows are represented by: a source of packets (a colored circle labelled by the flow index), a path (a sequence of arrows from the source, through the nodes), and a destination of the packets (a colored circle with dashed boundary labelled by the flow index). This example of network is used to illustrate an issue with dynamic deadlines in Section 3.1.

However, even if $\forall t \in [t_1, t_2]$, the linear combination $\mathbf{D}(t)$ always belongs to $\overline{\mathbb{D}}(\mathcal{G})$, the end-to-end deadline may be missed anyway. Suppose we have the network $\mathcal{G}$, illustrated in Figure 3 and focus on flow 1 (the blue flow), with path $p_1 = \{1, 2\}$. The end-to-end deadline for this flow is $\mathcal{D}_1 = 6$ milliseconds (ms). Suppose now that in response to an increase of the incoming packets rate, node 2 must change its deadline from $D_2(t_1) = 1$ to $D_2(t_2) = 5$. The node deadlines of the system are therefore changing from $\mathbf{D}(t_1) = [5, 1, \ldots]$ to $\mathbf{D}(t_2) = [1, 5, \ldots]$. Please, note that for the purpose of this example the particular choice of deadlines for nodes 3, 4, and 5 does not matter.

Figure 4 shows the deadlines node 1 and node 2 over time. At time $\tau_1 = 1$ a packet enters node 1, which has $D_1(\tau_1) = 5$. In the worst case, node 1 will finish processing that packet at time $\tau_1 + D_1(\tau_1) = 6$ms. Suppose now, that at time $t_1 = 2$ms, while this packet is still at node 1, the resource manager begins changing the node deadlines from $\mathbf{D}(t_1) = [5, 1, \ldots]$ towards $\mathbf{D}(t_2) = [1, 5, \ldots]$. When the packet exits node 1 at time $\tau_1 + D_1(\tau_1) = t_2 = 6$, then it enters node 2. Due to the change of node deadlines, now the value of $D_2(t)$ at $t = 6$ is $D_2(6) = 5$. This means that, in the worst case, the response time of the second node is also 5ms, since $R_2(t_2) \leq D_2(t_2) = 5$. The packet that entered flow 1 at time $\tau_1 = 1$ would therefore, in the worst case, may take up to 10ms to traverse its path p_1. Hence, the end-to-end deadline $\mathcal{D}_1 = 6$ of the packet is violated, despite the fact the sum of the node deadlines $D_1(t)$ and $D_2(t)$ along the path was never greater than $\mathcal{D}_1$ (that is $\forall t,\ D_1(t) + D_2(t) \leq \mathcal{D}_1$).

This simple example shows that when the network is allowed to change the node deadlines dynamically, the constraint $\mathbf{D}(t) \in \overline{\mathbb{D}}(\mathcal{G})$ is not a sufficient condition to guarantee end-to-end deadlines to be met. In fact, the violation of the end-to-end deadline illustrated by this example is related to the variation of the node deadlines. The node deadlines $\mathbf{D}(t)$ need to satisfy a stricter constraint, which is discussed next.

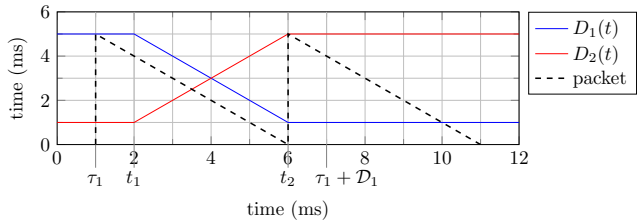


Figure 4 Example of how dynamic node deadlines may lead to end-to-end deadline violations. The node deadlines $D_1(t)$ (in blue) and $D_2(t)$ (in red) are changed from $\mathbf{D}(t_1) = [5, 1, \ldots]$ to $\mathbf{D}(t_2) = [1, 5, \ldots]$. This may lead a packet in flow 1 (with path $p_1 = \{1, 2\}$, as shown in Figure 3) to miss its end-to-end deadline $\mathcal{D}_1 = 6$.

3.2 Guaranteeing end-to-end deadlines

In this section, we provide the conditions that allow the network to dynamically change the node deadlines without incurring any end-to-end violation, as exemplified in Section 3.1. In fact, by letting node deadlines change over time, prediction of the end-to-end response time of packets becomes difficult. In Protocol 1 we present a solution to this problem. The intuition is that by limiting the rate-of-change of the node deadlines, it is possible to compute the end-to-end response time of a flow $j \in \mathcal{F}$ at time t. This allows us to compute the allowed node deadlines. This is formally proved in Theorem 6.

Protocol 1 Management of dynamic node deadlines in static networks.

- The node deadlines can never change with a rate larger than some fixed $\alpha \in [0,1]$:

  $$\forall t \geq 0,\ \forall i \in \mathcal{V},\ |\dot{D}_i(t)| \leq \alpha.$$

- The node deadlines must always be within the set of *feasible node deadlines*:

  $$\mathbf{D}(t) \in \mathbb{D}(\mathcal{G}) = \Big\{ D_i \in \mathbb{R}^+ : D_i \geq \underline{R}_i,\ i \in \mathcal{V},\ \forall j \in \mathcal{F},\ \sum_{i=1}^{\ell_j} (1+\alpha)^{\ell_j - i} D_{p_j(i)} \leq \mathcal{D}_j \Big\}.$$

▶ **Theorem 6** (Dynamic deadlines in static networks). *No end-to-end deadlines of any flow is violated, that is*

$$\forall t \geq 0,\ \forall j \in \mathcal{F} \quad \mathcal{R}_j(t) \leq \mathcal{D}_j, \tag{6}$$

as long as the node deadlines $\mathbf{D}(t)$ *never change with a rate faster than a given bound* $\alpha \in [0,1]$*:*

$$\forall t \geq 0,\ \forall i \in \mathcal{V}, \qquad |\dot{D}_i(t)| \leq \alpha, \tag{7}$$

and as long as the node deadlines remain within the space of feasible node deadlines:

$$\forall t \geq 0, \quad \mathbf{D}(t) \in \mathbb{D}(\mathcal{G}), \tag{8}$$

with $\mathbb{D}(\mathcal{G})$ *given by*

$$\mathbb{D}(\mathcal{G}) = \Big\{ D_i \in \mathbb{R}^+ : \forall i \in \mathcal{V},\ D_i \geq \underline{R}_i, \quad \forall j \in \mathcal{F},\ \sum_{i=1}^{\ell_j} (1+\alpha)^{\ell_j - i} D_{p_j(i)} \leq \mathcal{D}_j \Big\}. \tag{9}$$

Proof. We begin by recalling Assumption 1: the nodes in the network have a response-time controller which ensures that Eq. (1) always holds, that is $\forall t \geq 0,\ R_i(t) \leq D_i(t)$. It follows that for the first node $p_j(1)$ of any path $j \in \mathcal{F}$, it is always true that:

$$\forall t \geq 0,\ \forall j \in \mathcal{F}, \quad \mathcal{R}_{j,1}(t) = R_{p_j(1)}(t) \leq D_{p_j(1)}(t). \tag{10}$$

It then follows that for all subsequent nodes with index $i > 1$, from Eq. (3), we have that

$$\begin{aligned} \forall t \geq 0,\ \forall j \in \mathcal{F}, \quad \mathcal{R}_{j,i}(t) &= \mathcal{R}_{j,i-1}(t) + R_{p_j(i)}\big(t + \mathcal{R}_{j,i-1}(t)\big) \\ &\leq \mathcal{R}_{j,i-1}(t) + D_{p_j(i)}\big(t + \mathcal{R}_{j,i-1}(t)\big) \\ &\leq \mathcal{R}_{j,i-1}(t) + D_{p_j(i)}(t) + \alpha\, \mathcal{R}_{j,i-1}(t) \\ &= (1+\alpha)\mathcal{R}_{j,i-1}(t) + D_{p_j(i)}(t) \end{aligned} \tag{11}$$

where each step follows:
1. from the definition of end-to-end response-time (3),
2. from Eq. (1),
3. from Eq. (7), which implies Lipschitz-continuity of $D_i(t)$ with Lipschitz-constant α,
4. from basic math.

In the next step, we show

$$\forall t \geq 0,\ \forall j \in \mathcal{F}, \quad \forall x = 1, \ldots, \ell_j, \qquad \mathcal{R}_{j,x}(t) \leq \sum_{i=1}^{x} (1+\alpha)^{x-i} D_{p_j(i)}(t), \tag{12}$$

holds by proving by induction on the index x of nodes over the path j. When $x = 1$, Eq. (12) follows directly from (10). For any other $x > 1$, we have that

$$\begin{aligned}\mathcal{R}_{j,x}(t) &\leq (1+\alpha)\mathcal{R}_{j,x-1}(t) + D_{p_j(x)}(t) \\ &\leq (1+\alpha)\sum_{i=1}^{x-1}(1+\alpha)^{x-1-i}D_{p_j(i)}(t) + D_{p_j(x)}(t) \\ &= \sum_{i=1}^{x-1}(1+\alpha)^{x-i}D_{p_j(i)}(t) + D_{p_j(x)}(t) \\ &= \sum_{i=1}^{x}(1+\alpha)^{x-i}D_{p_j(i)}(t)\end{aligned}$$

where the different steps follow:

1. because the inequality is the same as Eq. (11)
2. because we exploit the inductive hypothesis of (12) for $x - 1$,
3. from basic math.

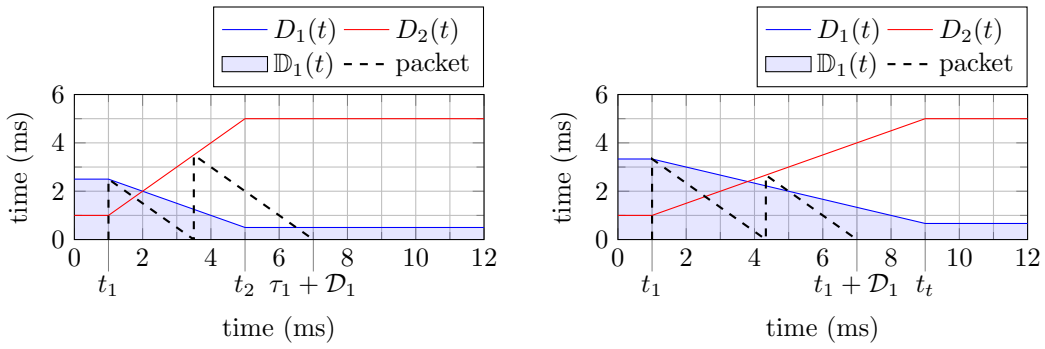
Hence, Eq. (12) is proved for all $x = 1, \ldots, \ell_j$.

Finally, we can conclude the proof by showing that

$$\forall t \geq 0,\ \forall j \in \mathcal{F} \qquad \mathcal{R}_j(t) = \mathcal{R}_{j,\ell_j}(t) \leq \sum_{i=1}^{\ell_j}(1+\alpha)^{\ell_j - i}D_{p_j(i)}(t) \leq \mathcal{D}_j$$

which implies that as long as the node deadlines are chosen such that $\mathbf{D}(t) \in \mathbf{D}(\mathcal{G})$ no end-to-end deadlines of any flow $j \in \mathcal{F}$ is violated, and the theorem is proved. ◀

3.3 Example – fixed by applying Theorem 6



(a) Maximum rate-of-change $\alpha = 1$.

(b) Maximum rate-of-change $\alpha = 0.5$.

Figure 5 Modified example from Section 3.1. We illustrate how the system dynamically changes node deadlines over time, with different rates of change. In 5a and 5b the nodes deadlines are allowed to change with a rate of $\alpha = 1$ and $\alpha = 0.5$, respectively. Given the change of $D_2(t)$ (illustrated in red) from 1 to 5, the resource manager is only allowed to change $D_1(t)$ ensuring that it remains within the shaded blue region (denoted by $\mathbb{D}_1(t)$ and representing the space of feasible node deadlines for $D_1(t)$). Finally, as illustrated by the dashed black line, by ensuring that $D_1(t)$ remains within the blue region, the longest time it will take a packet to traverse flow 1 will be less than its end-to-end deadline.

This section illustrates that if the node deadlines are changed in accordance to Protocol 1, then the issue of end-to-end deadline misses shown in Section 3.1 cannot happen. To do so, we modify the example of Section 3.1. We consider again flow 1 of Figure 3, which has path $p_1 = \{1, 2\}$ and end-to-end deadline $\mathcal{D}_1 = 6$.

In this scenario, we assume that the resource manager must change $D_2(t)$ from $D_2(t_1) = 1$ to $D_2(t_2) = 5$. The reason is to allow node 2 to handle a sudden increase in incoming traffic. Recalling the example in Section 3.1, we can now verify that if the node deadlines begin in a state of $D_1(t_1) = 5$ and $D_2(t_1) = 1$ the resource manager is not able to change them at all. In fact, by writing explicitly the constraint of Eq. (9) in Protocol 1 for the path $j = 1$, we have

$$D_1(t_1) + (1 + \alpha)D_2(t_1) \leq \mathcal{D}_1$$
$$1 + (1 + \alpha)5 \leq 6 \qquad \Rightarrow \qquad \alpha \leq 0$$

which means that node deadlines are not allowed to change at all. If some change is needed (for example, to handle a burst of incoming traffic), then the node deadlines must be more constrained. In fact, given the change of $D_2(t)$, the choice of $D_1(t)$ has to satisfy the following constraint:

$$\forall t \geq 0, \quad (1 + \alpha)D_1(t) + D_2(t) \leq \mathcal{D}_1 = 6. \tag{13}$$

Next, we compare two cases of different values of feasible rate of change α for the node deadlines. The result are illustrated in Figure 5.

Maximum rate-of-change $\alpha = 1$

To allow the network to transition quickly from $D_2(t_1) = 1$ to $D_2(t_2) = 5$ we choose $\alpha = 1$. This means that the the choices of D_1 are constrained by the following condition:

$$\forall t \geq 0, \quad (1 + 1)D_1(t) + D_2(t) \leq \mathcal{D}_1 = 6.$$

This condition gives the resource manager the space of possible choices for $D_1(t)$ illustrated by the shaded blue region in Figure 5a. The largest possible choices for $D_1(t)$ therefore involves changing D_1 from $D_1(t_1) = 2.5$ to $D_1(t_2) = 0.5$, with $t_1 = 1$ and $t_2 = 5$.

Maximum rate-of-change $\alpha = 0.5$

The stringency of the constraint on the node deadlines when $\alpha = 1$ may be relaxed by requiring a slower transition between the node deadlines, for example $\alpha = 0.5$. By doing so, we get the following conditions on $D_1(t)$:

$$D_1(t_1) \leq \frac{10}{3} \approx 3.333, \qquad D_1(t_2) \leq \frac{2}{3} \approx 0.666,$$

in order to ensure that Eq. (13) holds, and assuming that $D_2(t)$ is again changed from $D_2(t_1) = 1$ to $D_2(t_2) = 5$. Not surprisingly, by requiring a smoother transition, the node deadlines may be larger. Similarly to the previous case, this is illustrated in Figure 5b, where we illustrate that when $D_1(t)$ is chosen to be as large as possible, a packet traversing flow 1 is always guaranteed to meet its end-to-end deadline of 6ms.

Trade-offs with alpha

This example illustrates some fundamental trade-offs that come by adopting Protocol 1. While it allows the system to change the node deadlines dynamically with a rate of α, it imposes some constraints on the possible choices of node deadlines. The quicker one wishes to change the node deadlines, the more restricted the choice of feasible node deadlines becomes, and vice versa. The design-parameter α should be chosen appropriately for a given application.

4 Dynamic networks

In this section, we generalize the method presented Section 3 to the case of a *dynamic network* $\mathcal{G}(t) = \{\mathcal{V}(t),\ \mathcal{E}(t),\ \mathcal{F}(t)\}$, where nodes and flows may join and leave the network at run time. We begin by showing, in Corollary 7, that the result of Theorem 6 transfers directly to a dynamic network. This means that the end-to-end deadline of any flow in the network will be met as long as the hypothesis of Theorem 6 hold for all states of the network $\mathcal{G}(t)$ at all times t. Conditions stated in Corollary 7 are fulfilled. By comparing Theorem 6 and Corollary 7 one can see that the only difference is that we now allow for a dynamic network, i.e., $\mathcal{G}(t)$ instead of a static network $\mathcal{G}$.

▶ **Corollary 7** (Dynamic deadlines in dynamic networks)**.** *The end-to-end deadline of all flows are always met, at all times, that is:*

$$\forall t \geq 0,\ \forall j \in \mathcal{F}(t) \quad \mathcal{R}_j(t) \leq \mathcal{D}_j, \tag{14}$$

as long as the node deadlines $\mathbf{D}(t)$ *never change with a rate faster than some* $\alpha \in [0, 1]$*:*

$$\forall t \geq 0,\ \forall i \in \mathcal{V}(t), \qquad |\dot{D}_i(t)| \leq \alpha, \tag{15}$$

and as long as the node deadlines belong to the space of feasible node deadlines:

$$\forall t \geq 0, \quad \mathbf{D}(t) \in \mathbb{D}(\mathcal{G}(t)), \tag{16}$$

with $\mathbb{D}(\mathcal{G}(t))$ *given by*

$$\mathbb{D}\big(\mathcal{G}(t)\big) = \Big\{ D_i \in \mathbb{R}^+ : \forall i \in \mathcal{V}(t),\ D_i \geq \underline{R}_i, \quad \forall j \in \mathcal{F}(t),\ \sum_{i=1}^{\ell_j} (1+\alpha)^{\ell_j - i} D_{p_j(i)} \leq \mathcal{D}_j \Big\}. \tag{17}$$

Proof. We begin the proof by observing that from Eq. (15), it follows directly from Theorem 6 that for a fixed time-instance t' it holds that

$$\forall j \in \mathcal{F}(t'), \quad \mathcal{R}_j(t') \leq \mathcal{D}_j, \tag{18}$$

as long as

$$\mathbf{D}(t') \in \mathbb{D}\big(\mathcal{G}(t')\big), \tag{19}$$

with $\mathbb{D}\big(\mathcal{G}(t')\big)$ given by Eq. (17). In fact, this is precisely what was stated and proved in Theorem 6, but with a fixed network topology $\mathcal{G}$, instead of an instantaneous "snapshot" $\mathcal{G}(t')$ of a dynamic topology.

Then, the hypothesis of Eq. (16) ensures that Eq. (19) holds $\forall t \geq 0$. Therefore, it follows that Eq. (18) holds $\forall t \geq 0$, and in turn that Eq. (14) does always hold, as required. ◀

As demonstrated by the short proof, Corollary 7 does not poses any deeper conceptual challenges compared to Theorem 6. However, the two hypothesis of (15) and (16) may be hard to hold simultaneously, if no special care is taken. This is illustrated in the next example.

Example – issues in acceptance a new flow

The blind admission of new flows as soon as they request to join, may cause the violation of one of the two hypothesis of the corollary (Equations (15) and (16)) making then Corollary 7 incapable to guarantee end-to-end deadlines. At the time when any new flow j' is admitted

to the network, the set of flows $\mathcal{F}(t)$ includes the new flow j' which was not previously in the set. As a consequence of the acceptance of the new flow j' into $\mathcal{F}(t)$, the set of feasible node deadlines $\mathbb{D}\big(\mathcal{G}(t)\big)$ may suddenly shrink due to the newly added constraint. As illustrated in the next example, this might in turn cause the node deadlines to be in an infeasible state, such that the end-to-end deadline of the newly accepted flow will be violated. Notice that node deadlines **cannot** instantaneously adapt to the new constraint, otherwise the hypothesis of "bounded rate of change" of Eq. (15) is violated.

In Figure 6, we illustrate a system where a new flow j' registers to join the network at time $f_{j'}^{\mathsf{rq}+} = 3$. As soon as this flow is accepted into the network, at time $f_{j'}^{\mathsf{ok}+} = 4.5$, the set of feasible node deadlines (illustrated by the shaded green area) make a discrete jump. This means that the node deadlines $\mathbf{D}(t)$ (black thick line in Figure 6) will no longer remain within $\mathbb{D}\big(\mathcal{G}(t)\big)$. In other words this means that $\mathbf{D}(f_{j'}^{\mathsf{ok}+}) \notin \mathbb{D}\big(\mathcal{G}(f_{j'}^{\mathsf{ok}+})\big)$, which is a clear violation of the conditions of Corollary 7.

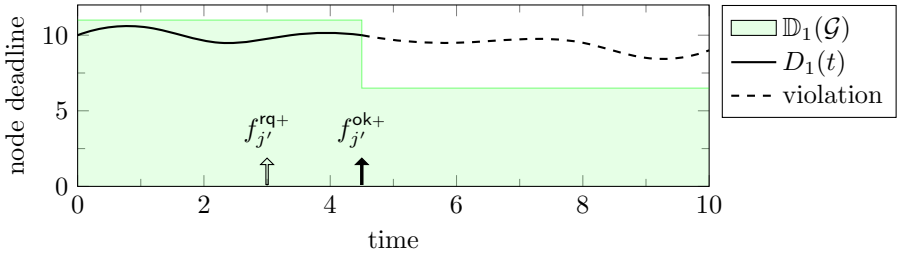


Figure 6 Illustration of why it might be difficult to ensure that $\forall t \geq 0$, $\mathbf{D}(t) \in \mathbb{D}\big(\mathcal{G}(t)\big)$ and $|\dot{D}_i(t)| \leq \alpha$. In this scenario, the space of feasible node deadline choices for the first node, $\mathbb{D}_1\big(\mathcal{G}(t)\big)$ (shaded green area), makes a discrete jump as soon as the new flow j' is accepted into the network. The reason is that the inclusion of the new flow adds a new end-to-end deadline constraints. This, in turn, may cause the node deadlines to instantaneouslt become infeasible."leave" the space of feasible node deadlines (illustrated by the dashed line).

The illustrated issue suggests that a special care must be taken when the network $\mathcal{G}(t)$ is modified, since the discrete variation of the network may not be compatible with the need of smooth node deadlines. Therefore, in Section 4.1, we present two protocols which address this issue:

- Protocol 2 explains how the requests of flows are managed, while
- Protocol 3 illustrates the management of nodes of the network.

4.1 Protocol allowing dynamic networks

In this section, we present a protocol for managing how flows may join and leave the network. We show that by following this protocol, the hypothesis of Corollary 7 do always hold, making then the corollary applicable. We then present a second protocol for how to manage nodes joining and leaving the network.

Management of flows

The intuition behind Protocol 2, which manages the flows, comes from the fact that as soon as a new flow j' is accepted into the network, at time $f_{j'}^{\mathsf{ok}+}$, the network changes from $\mathcal{G} = \{\mathcal{V}(t), \mathcal{E}(t), \mathcal{F}(t)\}$ to $\mathcal{G}^+ = \big\{\mathcal{V}(t), \mathcal{E}(t), \mathcal{F}(t) \cup \{j'\}\big\}$. The constraint corresponding to the new flow j' is thus added to $\mathbb{D}(\mathcal{G})$ leading to the new set of feasible node deadlines $\mathbb{D}(\mathcal{G}^+) \subseteq \mathbb{D}(\mathcal{G})$. Therefore, in order to ensure that the node deadlines remain within the set

of feasible node deadlines once j' is accepted, i.e., that $\mathbf{D}(f_{j'}^{\mathsf{ok}+}) \in \mathbb{D}\big(\mathcal{G}(f_{j'}^{\mathsf{ok}+})\big)$, Protocol 2 will only accept j' if $\mathbf{D}(t) \in \mathbb{D}(\mathcal{G}^+)$. If this is the case when j' requests to join, i.e., that $\mathbf{D}(f_{j'}^{\mathsf{rq}+}) \in \mathbb{D}(\mathcal{G}^+)$, then the new flow will be accepted immediately, and $f_{j'}^{\mathsf{ok}+} = f_{j'}^{\mathsf{rq}+}$.

If on the other hand, $\mathbf{D}(f_{j'}^{\mathsf{rq}+}) \notin \mathbb{D}(\mathcal{G}^+)$, it is not possible to accept j' immediately. Therefore, in order to accept it, the resource manager changes the node deadlines towards a *goal point* $\mathbf{D}^* \in \mathbb{D}(\mathcal{G}^+)$. It will then accept the new flow j', once $\mathbf{D}(t) = \mathbf{D}^*$, which will occur at time $f_{j'}^{\mathsf{ok}+}$, given by Eq. (20).

Protocol 2 Management of flows.

- At time $f_{j'}^{\mathsf{rq}+}$, the new flow j' requests to join
- If $\mathbf{D}(f_{j'}^{\mathsf{rq}+}) \in \mathbb{D}(\mathcal{G}^+)$, then the flow j' is admitted immediately, that is $f_{j'}^{\mathsf{ok}+} = f_{j'}^{\mathsf{rq}+}$.
- If $\mathbf{D}(f_{j'}^{\mathsf{rq}+}) \notin \mathbb{D}(\mathcal{G}^+)$, then the admission of flow j' is delayed until the node deadlines have completed a linear transition to a *goal point* $\mathbf{D}^* \in \mathbb{D}(\mathcal{G}^+)$, that is

$$f_{j'}^{\mathsf{ok}+} = f_{j'}^{\mathsf{rq}+} + \frac{\max_{i\in\mathcal{V}(t)} \left| D_i^* - D_i(f_{j'}^{\mathsf{rq}+}) \right|}{\alpha}. \tag{20}$$

  with α being the maximum feasible rate of change of node deadlines.
- If $\mathbf{D}\big(\mathcal{G}^+\big)\big) = \varnothing$, then the request of flow j' to join the network is rejected.
- A flow $j \in \mathcal{F}(t)$ may notify the resource manager and leave the network at any time. The time when it leaves is denoted by f_j^-.

Let us comment on the choice of the "goal point" $\mathbf{D}^*$. In general, any choice of $\mathbf{D}^* \in \mathbb{D}(\mathcal{G}^+)$ is valid. However, if the target is to minimize the transition-time fron the time $f_{j'}^{\mathsf{rq}+}$ flow j' request to join to the time $f_{j'}^{\mathsf{ok}+}$ it is admitted to the network (given by Eq. (20)), then it should be chosen as

$$\mathbf{D}^* = \operatorname*{arg\,min}_{\mathbf{D}\in\mathcal{G}^+} \max_i \left| D_i - D_i(f_{j'}^{\mathsf{rq}+}) \right|.$$

We would like to point out three observations from Protocol 2. The first one is that during the transition to accept the new flow j' the node deadlines are changed according to the following linear function:

$$\forall i \in \mathcal{V}(f_{j'}^{\mathsf{rq}+}),\ \forall t \in [f_{j'}^{\mathsf{rq}+}, f_{j'}^{\mathsf{ok}+}], \quad D_i(t) = \frac{D_i(f_{j'}^{\mathsf{rq}+}) \times (t - f_{j'}^{\mathsf{rq}+})}{f_{j'}^{\mathsf{ok}+} - f_{j'}^{\mathsf{rq}+}} + \frac{D_i^* \times (f_{j'}^{\mathsf{ok}+} - t)}{f_{j'}^{\mathsf{ok}+} - f_{j'}^{\mathsf{rq}+}}. \tag{21}$$

This means that the node deadlines are changed along a line from $\mathbf{D}(f_{j'}^{\mathsf{rq}+})$ to $\mathbf{D}(f_{j'}^{\mathsf{ok}+})$. Since $\mathbb{D}(\mathcal{G})$ is a convex space, it follows that $\mathbf{D}(t) \in \mathbb{D}(\mathcal{G})$ during the transition. Hence, it also follows that the hypothesis of Eq. (16) in Corollary 7 holds during the transition.

The second observation is that by changing the node deadlines according to Eq. (21) we acquire the property that $D_i(f_{j'}^{\mathsf{ok}+}) = D_i^*$. This means that at the end of the transition, at time $t = f_{j'}^{\mathsf{ok}+}$, we have $\mathbf{D}(t) \in \mathbb{D}(\mathcal{G}^+)$. This implies that once the new flow j' is accepted condition (16) of Corollary 7 continues to hold.

The final observation is that by exploiting the value of $f_{j'}^{\mathsf{ok}+}$ of Eq. (20), we have

$$|\dot{D}_i(t)| = \frac{|D_i(f_{j'}^{\mathsf{rq}+}) - D_i^*|}{f_{j'}^{\mathsf{ok}+} - f_{j'}^{\mathsf{rq}+}} = \alpha \frac{|D_i(f_{j'}^{\mathsf{rq}+}) - D_i^*|}{\max_{i\in\mathcal{V}(t)} \left| D_i^* - D_i(f_{j'}^{\mathsf{rq}+}) \right|} \leq \alpha \tag{22}$$

This implies that Protocol 2 fulfills condition (15) of Corollary 7, since the maximum rate-of-change of any node is α during the transition.

By combining all three observations, we can conclude that by following Protocol 2, the hypotheses of Corollary 7 are always met when accepting new flows, and then no end-to-end deadline is violated.

Finally, we remark that if there is no possible choice of a goal point, i.e., if $\mathbb{D}(\mathcal{G}^+) = \varnothing$, then the request of j' to join is clearly rejected because the admission of the new flow j' may cause the violation of end-to-end deadline of some flow already admitted to the network.

Management of nodes

The basic idea behind Protocol 3 is that when a node $i \in \mathcal{V}(t)$ leaves the network, it does affect the flows with a path going through i. Therefore, at time $v_i^{\mathsf{rq}-}$, that is when node i requests to leave the network, the resource manager notifies any affected flow $j \in \{j \ : \ \forall j \in \mathcal{F}(t),\ i \in p_j\}$ that it will be pushed out from the network after a time T_i^{leave}. The rationale for this is simply that the resource manager will no longer be able to provide any guarantees for the end-to-end deadlines of the affected flows once node i has left the network.

However, should the affected flows still wish to use some of the services in the network, they may request to re-join the network as a new flow. In order to allow the affected flows adequate time to do this, and to also ensure that there is enough time for the packets in the affected flows, we require T_i^{leave} to be greater than the largest end-to-end deadline of the affected flows.

Moreover, it can be noticed that there is no condition on the nodes willing to join the network.

Protocol 3 Management of nodes.

- A node $i \notin \mathcal{V}(t)$ may notify the resource manager and join the network at any time v_i^+.
- When a node $i \in \mathcal{V}(t)$ requests to leave the network (we denote such an instant by $v_i^{\mathsf{rq}-}$), it is allowed to do so at:

  $$v_i^{\mathsf{ok}-} \geq v_i^{\mathsf{rq}-} + T_i^{\text{leave}}$$

  with $T_i^{\text{leave}} \geq \max\{\mathcal{D}_j \ : \ \forall j \in \mathcal{F}(t),\ i \in p_j\}$
- The resource manager will notify all the affected flows $j \in \{j \ : \ \forall j \in \mathcal{F}(t),\ i \in p_j\}$ that they will be kicked out at time $t = v_i^{\mathsf{ok}-}$ if they have not left the network by then.

Comments on handling multiple flows

It should be noted that while Protocol 2 only treats the case where a single flow j' request to join, it is possible to allow multiple flows to request. The management of this scenario can be achieved by introducing a *request queue*, and then applying Protocol 2 for the request at the head of the queue. Once the request is served, the resource manager can then repeat for the new head-of-the-queue request until there are no more pending requests. This methodology would only incur in a heavier notation which we prefer not to add to lighten the presentation.

Moreover, as we show in the experiments of Section 5.2 a new flow can be accepted in matter of milliseconds, or even micro seconds. Therefore, given the application of cloud robotics, it is fair to assume that there will not be multiple flows requesting to join simultaneously. And should there be, introducing a request queue will not incur any significant delay for accepting new flow.

4.2 Example – dynamic network topology

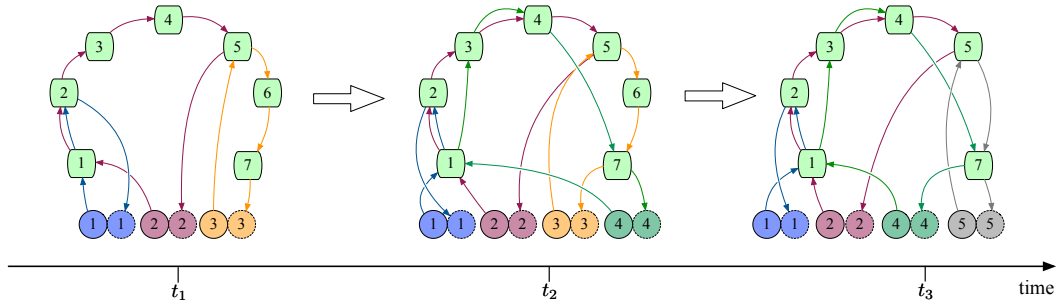


Figure 7 Illustration of how the network changes in the example of Section 4.2. In the first transition, from time t_1 to t_2, flow, 4 (green), joins the network, which already has flows 1, 2, and 3. Then in the second transition, from time t_2 to t_3, node 6 leaves the network. When node 6 leaves the network, it affects flow 3 (yellow), which then leaves, and re-join the network as a new flow 5 (gray).

In this section, by using some examples, we illustrate how the protocols between the flows, nodes, and the resource manager work.

The scenario is illustrated in Figure 7 and consists of the two transitions. In the first transition, from t_1 to t_2, flow 4 (green) requests to join the network. The second transition, from t_2 to t_3 illustrates how node 6 requests to leave the network. By doing so, it will affect flow 3, which has a path passing through node $i = 6$. The affected flow must therefore leave the network, and re-join as a new flow $j = 6$, illustrated by the gray arrows in Figure 7.

Next, we describe the first transition (of the new flow joining), followed by the second transition (node 6 leaving). The schedule for both transitions are depicted in Figure 8.

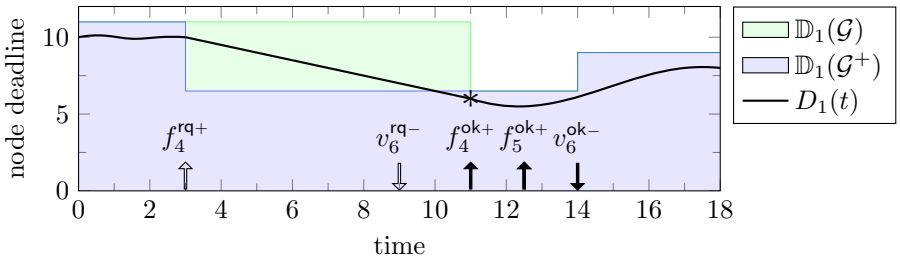


Figure 8 Illustration of how the space of feasible node deadlines changes when new flows are accepted into the network, as well as when nodes leave. It show a request from flow $j' = 4$ to join the network at time $f_4^{\mathsf{rq}+} = 3$ as well as a request from node $i = 6$ to leave the network at time $v_6^{\mathsf{rq}-} = 9$. It also illustrates how the resource manager changes the node deadlines towards $\mathbf{D}^*$ (given by $*$) before accepting the new flow $j' = 4$.

Request of a flow to join

When the new flow 4 requests to join the network, at time $f_4^{\mathsf{rq}+} = 3$, the node deadlines of the system are in a state such that it cannot be accepted immediately, i.e., $\mathbf{D}(f_4^{\mathsf{rq}+}) \notin \mathbb{D}(\mathcal{G}^+)$. According to Protocol 2, this requires the resource manager to change the node deadlines to a goal point $\mathbf{D}^* \in \mathbb{D}(\mathcal{G}^+)$. The node deadlines will therefore be changed linearly from $\mathbf{D}(f_4^{\mathsf{rq}+})$ to $\mathbf{D}^*$. At time $f_4^{\mathsf{ok}+}$, we have that $\mathbf{D}(t) = \mathbf{D}^*$, and then flow 4 is admitted into the network. This is illustrated in Figure 7 with $\mathbb{D}(\mathcal{G})$ shown as the shaded green region, $\mathbb{D}(\mathcal{G}^+)$ as the shaded blue region, and the goal point $\mathbf{D}^*$ as the $*$ symbol.

Request of a node to leave

At time $v_6^{\mathsf{rq}-} = 9$, node 6 requests to leave the network. This is illustrated in Figure 8 by the downward arrow. Since node 6 belongs to the path p_3 of flow 3 (see Figure 7), the departure of node 6 would affect flow 3 (yellow), with end-to-end deadline $\mathcal{D}_3 = 5$. By following Protocol 3, the resource manager will therefore notify flow 3 that it will no longer provide any end-to-end deadline guarantees after a time

$$v_6^{\mathsf{ok}-} = v_6^{\mathsf{rq}-} + T_6^{\text{leave}} = 9 + 5 = 14.$$

The node will then be allowed to leave the network at this time $v_6^{\mathsf{ok}-}$, as illustrated in Figure 8. In the figure, it can also be noticed that the space of feasible node deadlines increases when node 6 leaves. The reason is that constraint of the end-to-end deadline $\mathcal{D}_3$ of flow 3 is removed.

In this example, we assume that flow 3 still wants to remain in the network. Therefore, it will request to re-join the network as a new flow 5 at time $f_5^{\mathsf{rq}+}$. At this time, the node deadlines already allow the requesting flow 5 to join (that is $\mathbf{D}(f_5^{\mathsf{rq}+}) \in \mathbb{D}(\mathcal{G}^+)$) and then flow 5 is admitted immediately ($f_5^{\mathsf{ok}+} = f_5^{\mathsf{rq}+}$), as shown in Figure 8.

5 Evaluation: trade-offs with alpha

In this section we evaluate some of the effects introducing Protocols 1, 2, and 3 might have on a system. We are particularly concerned with the trade-offs of choosing different values of the design-parameter α. The intuition is that by choosing a value for α we choose how quickly the resource manager is able to change the node deadlines in the network. A higher value of α will therefore allow for quicker changes. This will in allow the resource manager to accept new flows faster. However, as illustrated in Section 3, a higher value of α will require lower node deadlines. This means that the response-times of the nodes in the network have to be lower. In order to satisfy this, some response-time controllers might have to sacrifice the quality of service (QoS) provided by the nodes. For instance, in [17] the sacrifice is to sometimes discard packets, and in [18] the sacrifice is to decrease the amount of content provided by the nodes.

5.1 System used for evaluation

The system used to evaluate the trade-offs of α is presented in a previous work [17] by the authors. In short, it allows nodes in a network, such as the one illustrated in Figure 9, to ensure that the response-time is less than a specific node deadline. The way it does this is by combining admission control and service control in every node.

The goal of the *service controller* is to ensure there is adequate processing capacity in the nodes. It does so by dynamically scaling the processing capacity according to a control-law. However, since the system is targeting a cloud-environment, the incoming traffic may be very dynamic. Moreover, the amount of processing capacity provided to the nodes may vary over time (i.e., servers might crash, etc.).

The goal of the *admission controller* is to always ensure that the response-time of the node is less than the node deadline. If there is sufficient processing capacity to serve the incoming traffic, it will not have to discard any packets. However, should a node find itself in a situation where there is not sufficient processing capacity to meet the incoming traffic, then the admission controller will have discard packets in order to guarantee that the response-time of the node will be less than the node deadline.

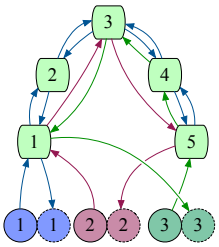


Figure 9 Illustration of the network used to evaluate the trade-offs with α and the time taken to reach the goal point $\mathbf{D}^*$ (in Section 5.2) as well as between α and the system performance (in Section 5.3).

5.2 Trade-off: alpha and time to accept a new flow

To evaluate the impact of α on the time required to accept a new flow, we will use the system presented in Section 5.1 and evaluate how long it takes the system to reach different goal points $\mathbf{D}^* \in \mathbb{D}(\mathcal{G}^+)$. Using a network with 5 nodes and 3 flows, we used the following set-up:

1. Choose the order-of-magnitude for the end-to-end deadlines $\overline{\mathcal{D}} \in \{0.1,\ 1,\ 10\}$ (milliseconds) as well as a value for $\alpha \in [10^{-3},\ 10^{0}]$.
2. Assign a randomly generated end-to-end deadline to each flow, $\mathcal{D}_j \in \mathcal{U}(0.7 \cdot \overline{\mathcal{D}},\ 1.3 \cdot \overline{\mathcal{D}})$. Note that $\mathcal{D}_j$ is drawn from a uniform distribution.
3. Chose the *goal point* $\mathbf{D}^*$ as the solution to the *optimal node-deadline problem*, presented in [17], but adapted with the constraints of Eq. (16):

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in \mathcal{V}(t)} 1/D_i \\ \text{subject to} \quad & \sum_{i=1}^{\ell_j} (1+\alpha)^{\ell_j - i} D_{p_j(i)} \leq \mathcal{D}_j \quad \forall j \in \mathcal{F}(t) \\ & D_i \geq 0 \qquad\qquad\qquad\qquad \forall i \in \mathcal{V}(t) \end{aligned} \tag{23}$$

4. Simulate how long it takes the network to reach the desired goal point $\mathbf{D}^*$.
5. Repeated steps 2 thru 4 for 100 simulations.
6. From the 100 simulations, compute the *average time* to reach a goal point $\mathbf{D}^*$.
7. Repeat steps 1 thru 6 for a different choices of $\overline{\mathcal{D}}$ and α.

The result of the evaluation is shown in Figure 10. As expected, it shows a clear relationship between α and the time needed to reach $\mathbf{D}^*$. It is interesting to note that even when having end-to-end deadlines in the order of 10ms, and a very small α the resource manager is still able to reach $\mathbf{D}^*$ in less than a second. By allowing a higher α, it is possible to reach $\mathbf{D}^*$ in less than a microsecond.

5.3 Trade-off: alpha and quality of service

To evaluate how different choices of α affect the QoS provided by the nodes in the network we will again use the system briefly presented in Section 5.1. As mentioned there, the system used an *admission controller* and a *service controller* to ensure that the response-time of the nodes always remained less than their node deadlines.

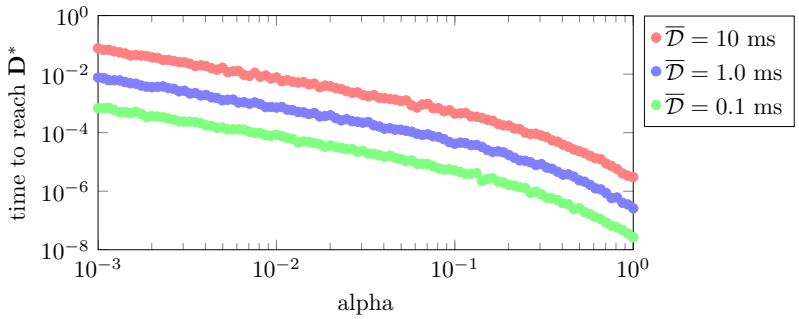


Figure 10 Simulation result of how long takes a system (with the network depicted in Figure 9) to reach a goal-point $\mathbf{D}^*$. It shows how this time depend on both the choice of α as well as how large the end-to-end deadlines of the systems are.

Due to the *uncertainties* of the available processing capacities, and since the amount of traffic going through the flows is highly varying, the admission controller sometimes have to discard packets. This is what we define as the QoS, in other words

$$\text{QoS} = 1 - \rho,$$

where ρ is the fraction of packets which are dropped.

The evaluation was performed using a network with 5 nodes and 3 flows, together with the following set-up:

1. Choose a *processing uncertainty* $\bar{\xi} \in \{0.05,\ 0.1,\ 0.2,\ 0.4\}$ and a value of $\alpha \in [10^{-3},\ 10^{0}]$..
2. Generate traffic based on data from the Swedish University Network (SUNET) and simulate the system for 20 seconds. A typical traffic pattern is illustrated in Figure 11a.
3. Compute the QoS for the simulation.
4. Repeat steps 2 and 3 for another 100 simulations.
5. Compute the *average* QoS for this choice of $\bar{\xi}$ and α.
6. Repeat steps 1 thru 5 for a new choice of $\bar{\xi}$ and α.

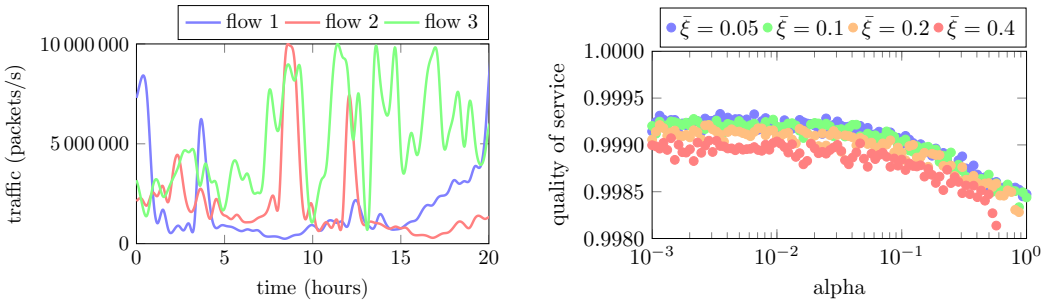
Some comments on the processing uncertainty above is that if $\bar{\xi} = 0.2$ a node in the network might believe it has a processing capacity of 1000 packets per second (pps), but in reality it could only handle 800 pps. Therefore, the higher $\bar{\xi}$, the higher the probability is that the node has a lack of available processing capacity.

The result of the evaluation, presented in Figure 11b, where the y-axis show the quality of service and the x-axis show the values of α. The different colors highlight the different bounds on the uncertainty for the processing capacity. An interesting observation is that the QoS does not depend so much on the uncertainty $\bar{\xi}$ as it does on the choice of α. As expected, when α increases, the QoS goes down. However, even for large values of α, the QoS remains fairly high, i.e., above 0.9980. This means that 99.8% of all the packets make it through the system on time.

6 Related work

Despite the fact that the addressed problem in this paper comes from very recent technology advancements (e.g. cloud computing and 5G), it is possible to abstract it in a way where related results can be found over quite vast a spectrum of older contributions. In an abstract way, the problem presented in this paper can be decomposed into the following sub-components:

- providing end-to-end deadline guarantees for flows in a network,
- splitting end-to-end deadlines into local deadlines.



(a) Traffic for one of the simulations.

(b) Evaluation of α and the quality-of-service.

Figure 11 Simulation to evaluate how α affect QoS of the system. It highlights that despite a highly varying traffic going through the flows (as depicted in Figure 11a) the QoS remains high, even for a value of α close to 1. In fact, it shows that for $\alpha = 0.1$ about 99.8% of all the packets made it through the system and met their end-to-end deadlines.

However, to the best of our knowledge, no work has considered all of these sub-components together in the context of a dynamic network topology.

A considerable amount of previous works addresses the deadline guarantee of a sequence of jobs that needs to be processed at a given node of a network [24, 20, 21]. Within each node, jobs are scheduled by any single processor scheduling policy (FP, EDF, or else). The communication between nodes is modelled by propagating the jitter [24, 20] or the offset [21] of the task execution within a node. Gerber et al. [5] proposed an alternate method to translate end-to-end deadlines over a directed graph of nodes into constraints on the activation periods of the tasks running at the intermediate nodes.

In the context of compositional analysis, previous works have addressed the problem of isolating and composing a single flow over a network of nodes. Lorente et al. [14] extended the holistic analysis to the case with nodes running at a fraction of computing capacity (abstracted by a bounded-delay time partition with bandwidth and delay). Jayachandran and Abdelzaher [10] developed several transformations ("delay composition algebra") to reduce the analysis of a distributed system to the single processor case. Serreli et al. [22] proposed a component interface for chains of tasks activated sporadically and an intermediate deadline assignment, which minimises the requested computing capacity. Similarly, Ashjaei et al. [1] proposed resource reservation over each node along the path.

In the context of computation happening at "small" scale, it is worth mentioning the modular analysis by Hamann, Jersak, Richter, Ernst [6]. Such a modular analysis, which found an application in the automotive domain, may well be a source of inspiration to analyze the schedulability within each node and the interaction between nodes. It is, however, orthogonal to our method which focuses on the policies to allow new flows of packets ("event streams" in the terminology of [6]) to be admitted at run time. Also, network calculus [13] and real-time calculus [23] are excellent orthogonal methods to analyze the schedulability within nodes as well as theier interactions.

The idea of breaking end-to-end deadlines in local deadlines was also exploited by several authors. Di Natale and Stankovic [2] proposed to split the end-to-end deadline proportionally to the local computation time or to divide equally the slack time. Marinca et al. [15] proposed two methods to assign local deadlines ("Fair Laxity Distribution" and "Unfair Laxity Distribution") to balance the distribution of the slack among the flows. Later, Jiang [11] used time slices to decouple the schedulability analysis of each node, reducing the complexity of the analysis. More recently, Hong et al. [8] formulated the local deadline assignment problem as a Mixed-Integer Linear Program (MILP) with the goal of maximising

the slack time. The number of local deadlines, however, is very high and makes the resulting optimisation problem hard to solve. Jabob et al. [9] proposed to split among local deadlines by using a *deadline ratio* $\rho \in (0, 1)$ configuration parameter chosen at design-time.

Related, but orthogonal to the presented research is the problem of mapping the flows of packets onto the available processing nodes. In the automotive context, Zhu et al. [26] formulated a MILP problem to find a task mapping that minimises the sum of a set of sensitive latencies. Garibay-Martínez et al. [4] used heuristics to partition tasks and assigned priorities to tasks sharing the same resource.

7 Conclusion and future works

In this work, we presented a framework, which allows applications and cloud-services to dynamically join and leave a system over time. The intuition is that by assigning and controlling how quickly local deadlines of cloud-services may change, it is possible to guarantee the end-to-end deadlines of the applications in presence of flows and nodes dynamically leaving and joining the network. Finally, with extensive simulations we are able to show that with the suggested protocols it is possible to accept new applications in matter of milliseconds. Moreover, we show that the constraints of the protocols does not affect the quality-of-service in a significant way.

This preliminary work opens for many research directions. Among them we mention:

impact of node policies How can the end-to-end response time benefit from per-flow packet scheduling policies within the nodes? For example, fixed priorities, EDF, etc.

decentralized protocol Can the framework be implemented in a decentralized way by exploiting per-node information rather than using the full knowledge, as in this paper?

References

1 Mohammad Ashjaei, Saad Mubeen, Moris Behnam, Luis Almeida, and Thomas Nolte. End-to-End Resource Reservations in Distributed Embedded Systems. In *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11, August 2016.

2 Marco Di Natale and John A. Stankovic. Dynamic End-to-end Guarantees in Distributed Real Time Systems. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, pages 215–227, December 1994.

3 Neha Gandhi, Dawn M Tilbury, Yixin Diao, J Hellerstein, and Sujay Parekh. Mimo control of an apache web server: Modeling and controller design. In *American Control Conference, 2002. Proceedings of the 2002*, volume 6, pages 4922–4927. IEEE, 2002.

4 Ricardo Garibay-Martínez, Geoffrey Nelissen, Luis Lino Ferreira, and Luis Miguel Pinho. Task partitioning and priority assignment for distributed hard real-time systems. *Journal of Computer and System Sciences*, 81(8):1542–1555, 2015.

5 Richard Gerber, Seongsoo Hong, and Manas Saksena. Guaranteeing Real-Time Requirements with Resource-based Calibration of Periodic Processes. *IEEE Transaction on Software Engineering*, 21(7):579–592, July 1995.

6 Arne Hamann, Marek Jersak, Kai Richter, and Rolf Ernst. A framework for modular analysis and exploration of heterogeneous embedded systems. *Real-Time Systems*, 33(1):101–137, July 2006. doi:10.1007/s11241-006-6884-x.

7 Dan Henriksson, Ying Lu, and Tarek Abdelzaher. Improved prediction for web server delay control. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 61–68. IEEE, 2004.

8 Shengyan Hong, Thidapat Chantem, and Xiaobo Sharon Hu. Local-deadline assignment for distributed real-time systems. *IEEE Transactions on Computers*, 64(7):1983–1997, 2015. doi:10.1109/TC.2014.2349494.

9 Romain Jacob, Marco Zimmerling, Pengcheng Huang, Jan Beutel, and Lothar Thiele. End-to-end real-time guarantees in wireless cyber-physical systems. In *Proceedings 2016 IEEE Real-Time Systems Symposium. RTSS 2016*, pages 167–178. IEEE, 2016.

10 Praveen Jayachandran and Tarek Abdelzaher. Delay Composition Algebra: A Reduction-Based Schedulability Algebra for Distributed Real-Time Systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 259–269, December 2008. doi:10.1109/RTSS.2008.38.

11 Shengbing Jiang. A decoupled scheduling approach for distributed real-time embedded automotive systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 191–198, 2006.

12 Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*, pages 700–711. ACM, 2014.

13 Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: a theory of deterministic queuing systems for the internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

14 José L. Lorente, Giuseppe Lipari, and Enrico Bini. A hierarchical scheduling model for component-based real-time systems. In *Proc. of the 20th International Parallel and Distributed Processing Symp.*, April 2006. doi:10.1109/IPDPS.2006.1639405.

15 Dana Marinca, Pascale Minet, and Laurent George. Analysis of deadline assignment methods in distributed real-time systems. *Computer Communications*, 27(15):1412–1423, 2004.

16 Victor Millnert, Johan Eker, and Enrico Bini. Dynamic control of NFV forwarding graphs with end-to-end deadline constraints. In *IEEE International Conference on Communications*, pages 1–7. IEEE, 2017.

17 Victor Millnert, Johan Eker, and Enrico Bini. Achieving predictable and low end-to-end latency for a network of smart services. In *IEEE GLOBECOM 2018*, 2018.

18 Tommi Nylander, Marcus Thelander Andrén, Karl-Erik Årzén, and Martina Maggio. Cloud Application Predictability through Integrated Load-Balancing and Service Time Control. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*, pages 51–60. IEEE, 2018.

19 Pradeep Padala, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 289–302. ACM, 2007.

20 José Carlos Palencia and Michael González Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, December 1998.

21 Rodolfo Pellizzoni and Giuseppe Lipari. Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling. *Journal of Computer and System Sciences*, 73(2):186–206, March 2007. doi:10.1016/j.jcss.2006.04.002.

22 Nicola Serreli, Giuseppe Lipari, and Enrico Bini. The Demand Bound Function Interface of Distributed Sporadic Pipelines of Tasks Scheduled by EDF. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, pages 187–196, July 2010. doi:10.1109/ECRTS.2010.17.

23 Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000.

24 Ken Tindell and John Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. *Microprocessing and Microprogramming*, 50:117–134, April 1994.

25 Thiemo Voigt and Per Gunningberg. Adaptive resource-based Web server admission control. In *ISCC*, 2002.

26 Qi Zhu, Haibo Zeng, Wei Zheng, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. Optimization of Task Allocation and Priority Assignment in Hard Real-time Distributed Systems. *ACM Transactions on Embedded Computing Systems*, 11(4):85:1–85:30, January 2013. doi:10.1145/2362336.2362352.