View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Institutional Research Information System University of Turin

Feedback for increased robustness of forwarding graphs in the cloud

Victor Millnert*, Johan Eker*†, Enrico Bini‡
*Lund University, Sweden
†Ericsson Research, Sweden
‡University of Turin, Italy

***Abstract*—Cloud computing technology provides the means to share physical resources among multiple users and data center tenants by exposing them as virtual resources. There is a strong industrial drive to use similar technology and concepts to provide timing sensitive services. One such domain is a chain of connected virtual network functions. This allows the capacity of each function to be scaled up and down by adding or removing virtual resources. In this work, we develop a model of a such a service chain and pose the dynamic allocation of resources as an optimization problem. We design and present a set of strategies to allot virtual network nodes in an optimal fashion subject to latency and buffer constraints. Furthermore, we derive a feedback-law for dynamically adjusting the amount of resources given to each functions in order to ensure that the system remains in the desired state even if there are modeling error or for a stochastic input.**

1. Introduction

Over the last years, cloud computing has swiftly transformed the IT infrastructure landscape, leading to large cost-savings for deployment of a wide range of IT applications. Some main characteristics of cloud computing are resource pooling, elasticity, and metering. Physical resources such as compute nodes, storage nodes, and network fabrics are shared among tenants. Virtual resource elasticity brings the ability to dynamically change the amount of allocated resources, for example as a function of workload or cost. Resource usage is metered and in most pricing models the tenant only pays for the allocated capacity.

While cloud technology initially was mostly used for IT applications, e.g. web servers, databases, etc., it is rapidly finding its way into new domains. One such domain is processing of network packages. Today network services are packaged as physical appliances that are connected together using physical network. Network services consist of interconnected network functions (NF). Examples of network functions are firewalls, deep packet inspections, transcoding, etc. A recent initiative from the standardisation body ETSI (European Telecommunications Standards Institute) addresses the standardisation of virtual network services under the name Network Functions Virtualisation (NFV) [1]. The expected benefits from this are, among others, better hardware utilisation and more flexibility, which translate into reduced capital and operating expenses (CAPEX and OPEX). A number of interesting use cases are found in [2], and in this paper we are investigating the one referred to as Virtual Network Functions Forwarding Graphs, see Figure 1.

We investigate the allocation of virtual resources to a given packet flow, i.e. what is the most cost efficient way

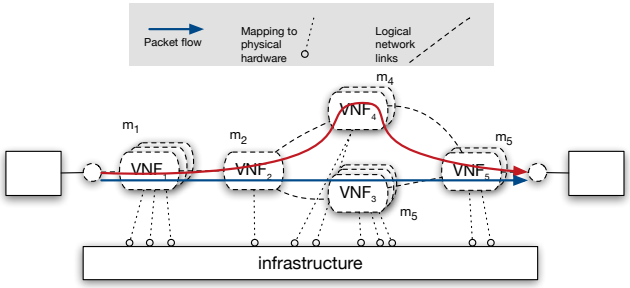

Figure 1: Several virtual networking functions (VNF) are connected together to provide a set of services. Packet flow through a specific path the VNFs (a virtual forwarding graph). The VNFs consist of a set of virtual resources, e.g., VNF_1 consist of three virtual machines, that are mapped onto physical hardware referred to as the virtual network function virtualization infrastructure (NFVI). In the figure there are two packet flows: a blue $\{\mathrm{VNF}_1, \mathrm{VNF}_2, \mathrm{VNF}_3, \mathrm{VNF}_5\}$, and a red $\{\mathrm{VNF}_1, \mathrm{VNF}_2, \mathrm{VNF}_4, \mathrm{VNF}_5\}$.

to allocate VNFs with a given capacity that still provide a network service within a given latency bound? The distilled problem is illustrated as the packet flows in Figure 1. The forwarding graph is implemented as a chain of virtual network nodes, also known as a service chains. To ensure that the capacity of a service chain matches the time-varying load, the number of instances m_i of each individual network function VNF_i may be scaled up or down.

The contribution of the paper is

- a mathematical model of the virtual resources supporting the packet flows in Figure 1,
- the set-up of an optimization problem for controlling the amount of resources needed by each function in the service chain,
- solution of the optimization-problem under the assumption of a constant input flow,
- a feedback-law for dynamically changing the resources used by each function, allowing for a stochastic input and impulse disturbances.

Related work

There are a number of well known and established resource management frameworks for data centers, but few of them explicitly address the issue of latency. Sparrow [3] presents an approach for scheduling a large number of

parallel jobs with short deadlines. The problem domain is different compared to our work in that we focus on sequential rather than parallel jobs. Chronos [4] focuses on reducing latency on the communication stack. RT-OpenStack [5] adds real-time performance to OpenStack by usage of a real-time hypervisor and a timing-aware VM-to-host mapping.

The enforcement of an end-to-end (E2E) deadline of a sequence of jobs to be executed through a sequence of computing elements was addressed by several works, possibly under different terminologies. In the holistic analysis [6], [7], [8] the schedulability analysis is performed locally. At global level the local response times are transformed into jitter or offset constraints for the subsequent tasks.

A second approach to guarantee an E2E deadline is to split a constraint into several local deadline constraints. While this approach avoids the iteration of the analysis, it requires an effective splitting method. Di Natale and Stankovic [9] proposed to split the E2E deadline proportionally to the local computation time or to divide equally the slack time. Later, Jiang [10] used time slices to decouple the schedulability analysis of each node, reducing the complexity of the analysis. Such an approach improves the robustness of the schedule, and allows to analyse each pipeline in isolation. Serreli et al. [11], [12] proposed to assign local deadlines to minimize a linear upper bound of the resulting local demand bound functions. More recently, Hong et al [13] formulated the local deadline assignment problem as a Mixed-Integer Linear Program (MILP) with the goal of maximizing the slack time. After local deadlines are assigned, the processor demand criterion was used to analyze distributed real-time pipelines [14], [12].

In all the mentioned works, jobs have non-negligible execution times. Hence, their delay is caused by the preemption experienced at each function. In our context, which is scheduling of virtual network services, jobs are executed non-preemptively and in FIFO order. Hence, the impact of the local computation onto the E2E delay of a request is minor compared to the queueing delay. This type of delay is intensively investigated in the networking community in the broad area *queuing systems* [15]. In this area, Henriksson et al. [16] proposed a feedforward/feedback controller to adjust the processing speed to match a given delay target.

Most of the works in queuing theory assumes a stochastic (usually markovian) model of job arrivals and service times. A solid contribution to the theory of deterministic queuing systems is due to Baccelli et al. [17], Cruz [18], and Parekh & Gallager [19]. These results built the foundation for the *network calculus* [20], later applied to real-time systems in the *real-time calculus* [21]. The advantage of network/real-time calculus is that, together with an analysis of the E2E delays, the sizes of the queues are also modelled. As in the cloud computing scenario the impact of the queue is very relevant since that is part of the resource usage which we aim to minimize, hence we follow this type of modeling.

2. Problem formulation

Section 1. We consider a *service-chain* consisting of n *functions* $F_1, \ldots, F_n$, as illustrated in Figure 2. Packets are flowing through the service-chain and they must be processed by each function in the chain within some end-to-end deadline, denoted by $D^{\max}$. A *fluid model* is used to approximate the packet flow and at time t there are $r_i(t) \in \mathbb{R}^+$ packets per second (pps) entering the i'th function and the the *cumulative arrived requests* for this function is

$$R_i(t) = \int_0^t r_i(\tau)\, d\tau. \tag{1}$$

In a recent benchmarking study it was shown that a typical virtual machine can process around 0.1–2.8 million packets per second, [22]. Hence, in this work the number of packets flowing through the functions is assumed to be in the order of millions of packets per second, supporting the use of a fluid model.

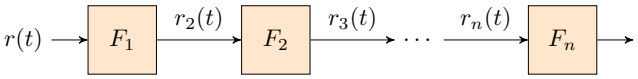

Figure 2: Illustration of the service-chain.

2.1. Service model

As illustrated in Figure 3, the incoming requests to function F_i are stored in the buffer and then processed once it reaches the head of the queue. At time t there are $m_i(t) \in \mathbb{Z}^+$ machines ready to serve the requests, each with a *nominal speed* of $\bar{s}_i \in \mathbb{R}^+$ (note that this nominal speed might differ between different functions in the service chain, i.e. it does not in general hold that $\bar{s}_i = \bar{s}_j$ for $i \neq j$). The *maximum speed* that function F_i can process requests at is thus $m_i(t)\bar{s}_i$. The rate by which F_i is actually processing requests at time t is denoted $s_i(t) \in \mathbb{R}^+$. The *cumulative served requests* is defined as

$$S_i(t) = \int_0^t s_i(\tau)\, d\tau. \tag{2}$$

At time t the number of requests stored in the queue is defined as the *queue length* $q_i(t) \in \mathbb{R}^+$:

$$q_i(t) = \int_0^t (r_i(\tau) - s_i(\tau))d\tau = R_i(t) - S_i(t). \tag{3}$$

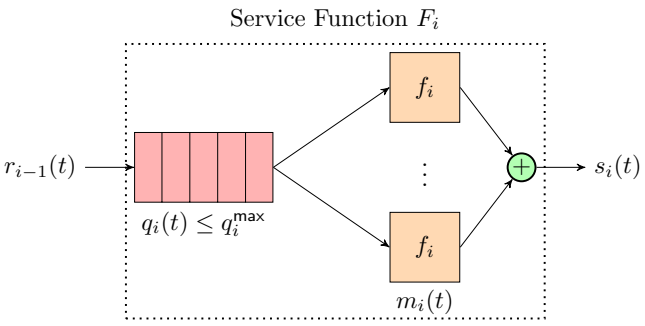

Figure 3: Illustration of the structure and different entities of the service chain.

Each function has a fixed *maximum-queue capacity* $q_i^{\max} \in \mathbb{R}^+$, representing the largest number of requests that can be stored at the function F_i.

The *queueing delay*, depends on the status of the queue as well as on the service rate. We denote by $D_{i,j}(t)$ the time taken by a request from when it enters function F_i to when it exits F_j, with $j \geq i$, where t is the time when the request exits function F_j:

$$D_{i,j}(t) = \inf\{\tau \geq 0 \,:\, R_i(t-\tau) \leq S_j(t)\}.$$

The *maximum queueing delay* then is $\hat{D}_{i,j} = \max_{t\geq 0} D_{i,j}(t)$. The requirement that a requests meets it end-to-end deadline is $\hat{D}_{1,n} \leq D^{\max}$.

To control the queueing delay, it is necessary to control the service rate of the function. Therefore, we assume that it is possible to change the maximum service-rate of a function by changing the number of machines that are on, i.e. changing $m_i(t)$. However, turning on a machine takes Δ_i^{on} time units, and turning off a machine takes Δ_i^{off} time units. Together they account for a *time overhead*, $\Delta_i = \Delta_i^{\mathsf{on}} + \Delta_i^{\mathsf{off}}$, associated with turning on/off a machine.

In 2012 Google profiled where the latency in a data center occurred, [4]. They showed that less than 1% ($\approx 1\mu s$) of the latency occurred was due to the propagation in the network fabric. The other 99% ($\approx 85\mu s$) occurred somewhere in the kernel, the switches, the memory, or the application. Since it is difficult to say exactly which of this 99% is due to processing, or queueing, we make the abstraction of considering queueing delay and processing delay together, simply as queueing delay. Furthermore, we assume that no request is lost in the communication links, and that there is no propagation delay. Hence the concatenation of the functions F_1 through F_n implies that the input of function F_i is exactly the output of function F_{i-1}, for $i = 2, \ldots, n$, as illustrated in Figure 2.

2.2. Cost model

To be able to provide guarantees about the behaviour of the service chain, it is necessary to make *hard reservations* of the resources needed by each function in the chain. This means that when a certain resource is reserved, it is guaranteed to be available for utilization. Reserving this resource results in a cost, and due to the hard reservation, the cost does not dependent on the actual utilisation, but only on the resource reserved.

The *computation cost* per time-unit per machine is denoted $\jmath_i^{\mathsf{c}}$, and can be seen as the cost for the CPU-cycles needed by one machine in F_i. This cost will also occur during the time-overhead Δ_i. Without being too conservative, this time-overhead can be assumed to occur only when a machine is started. The *average computing cost* per time-unit for the whole function F_i is then

$$J_i^{\mathsf{c}}(m_i(t)) = \lim_{t\to\infty} \frac{\jmath_i^{\mathsf{c}}}{t} \int_0^t m_i(s) + \Delta_i \cdot (\partial^- m_i(s))_+ ds \quad (4)$$

where $(x)_+ = \max(x, 0)$, and $\partial^- m_i(t)$ is the left-limit of $m_i(t)$:

$$\partial^- m_i(t) = \lim_{a\to t^-} \frac{m_i(t) - m_i(a)}{t - a},$$

that is, a sequence of Dirac's deltas at all points where the number of machines changes. This means that the value of the left-limit of $m_i(t)$ is only adding to the computation-cost whenever it is positive, i.e. when a machine is switched on.

The *queue cost* per time-unit per space for a request is denoted $\jmath_i^{\mathsf{q}}$. This can be seen as the cost that comes from the fact that physical storage needs to be reserved such that a queue can be hosted on it, normally this would correspond to the RAM of the network-card. Reserving the capacity of $q_i^{\max}$ would thus result in a cost per time-unit of

$$J_i^{\mathsf{q}}(q_i^{\max}) = \jmath_i^{\mathsf{q}} q_i^{\max}. \quad (5)$$

2.3. Problem definition

The aim of this paper is to control the number $m_i(t)$ of machines running in function F_i, such that the total average cost is minimized, while the E2E constraint $D^{\max}$ is not violated and the maximum queue sizes $q_i^{\max}$ are not exceeded. This can be posed as the following problem:

$$\begin{aligned} \text{minimize } & J = \sum_{i=1}^{n} J_i^{\mathsf{c}}(m_i(t)) + J_i^{\mathsf{q}}(q_i^{\max}) \\ \text{subject to } & \hat{D}_{1,n} \leq D^{\max} \\ & q_i(t) \leq q_i^{\max}, \quad \forall t \geq 0, \quad i = 1, 2, \ldots, n \end{aligned} \quad (6)$$

with J_i^{c} and J_i^{q} as in (4) and (5), respectively. In this paper the optimization problem (6) will be solved for a service-chain fed with a constant incoming rate r. Later on a feedback-law will be derived allowing the system to remain stable under a stochastic input.

A valid lower bound J^{lb} to the cost achieved by any feasible solution of (6) is found by assuming that all functions are capable of providing exactly a service rate r equal to the input rate. This is possible by running a fractional number of machines $r/\bar{s}_i$ at function F_i. In such an ideal case, buffers can be of zero size ($\forall i,\ q_i^{\max} = 0$), and there is no queueing delay ($\hat{D}_{1,n} = 0$) since service and the arrival rates are the same at all functions. Hence, the lower bound to the cost is

$$J^{\mathsf{lb}} = \sum_{i=1}^{n} \jmath_i^{\mathsf{c}} \frac{r}{\bar{s}_i}. \quad (7)$$

Such a lower bound will be used to compare the quality of the solution found later on.

3. Controlling the machines

In presence of an incoming flow of requests at a constant rate $r(t) = r$ packets per second, a number

$$\bar{m}_i = \left\lfloor \frac{r}{\bar{s}_i} \right\rfloor \quad (8)$$

of machines running in function F_i must always stay on. To match the incoming rate r, in addition to the $\bar{m}_i$ machines

always on, an additional machine must be on for some time in order to process a request rate of $\bar{s}_i\rho_i$ where ρ_i is the *normalized residual request* rate:

$$\rho_i = {}^{r}/_{\bar{s}_i} - \bar{m}_i, \qquad \rho_i \in [0,1). \tag{9}$$

Naturally, every function can decide arbitrarily when, and for how long, its additional machine is on as long as the average service rate for the function matches in incoming request rate r, and the queue does not remain zero for a prolonged amount of time. However, such an approach would lead to the complexity of the schedule blowing up exponentially with the number of functions in the chain, making the synthesis of a machine-schedule intractable. Hence, to reduce the complexity and make the analysis tractable the extra machines are restricted to be turned on/off with a global period T, i.e. the period by which every function switches on their extra machine is $T_i = T$. This implies that during every period the additional machine of the i'th function has to process the *residual work* of $T\times(r-\bar{m}_i\bar{s}_i)$. The necessary *on-time* T_i^{on}, needed by the extra machine, to process this work during the period is $T \times (r - \bar{m}_i\bar{s}_i)/\bar{s}_i = T\rho_i$. The remaining time of the period the additional machine should be switched off, denoted by T_i^{off}, leading to

$$T_i^{\mathsf{on}} = T\rho_i, \quad T_i^{\mathsf{off}} = T - T_i^{\mathsf{on}} = T(1-\rho_i). \tag{10}$$

Notice, however, that the actual time the extra machine is consuming power is $T_i^{\mathsf{on}} + \Delta_i$ due to the time-overhead for starting a new machine.

The design variable of the optimization problem (6) is now the period T, so it remains to investigate how it affects the computing-cost, the buffer-cost, and the end-to-end deadline.

The computing-cost is straightforward to find when the additional machines are switched on/off with a period T. If $\bar{m}_i+1$ machines are on for a time T_i^{on}, and only $\bar{m}_i$ machines are on for a time T_i^{off}, the cost J_i^{c} of (4) becomes:

$$J_i^{\mathsf{c}}(T) = \jmath_i^{\mathsf{c}}\left(\frac{T_i^{\mathsf{on}}+\Delta_i}{T} + \bar{m}_i\right) = \jmath_i^{\mathsf{c}}\left(\bar{m}_i + \rho_i + \frac{\Delta_i}{T}\right) \tag{11}$$

as long as $T_i^{\mathsf{off}} \geq \Delta_i$. If instead $T_i^{\mathsf{off}} < \Delta_i$, that is if

$$T < \overline{T} := \frac{\Delta_i}{1-\rho_i}, \tag{12}$$

there is no time to switch the additional machine off and then on again before the new period start. Hence, we keep the last machine on, even if it is not processing packets, and the computing cost depends on the period according to:

$$J_i^{\mathsf{c}}(T) = \jmath_i^{\mathsf{c}}\left(\bar{m}_i + \rho_i + \frac{T_i^{\mathsf{off}}}{T}\right) = \jmath_i^{\mathsf{c}}(\bar{m}_i+1). \tag{13}$$

It then remains to find the relationship between the period and the maximum queue-length—which by equation (5) translates to the buffer-cost—of the functions as well as to the maximum end-to-end delay. In Lemma 1 below, it is shown that the maximum queue-length is proportional to the period, and similarly, in Lemma 2 it is shown that the maximum end-to-end delay is also proportional to the period. The intuition behind this fact is that the longer the period T is, the longer a function will have to wait with the additional machine being off, before turning it on again. During this interval of time, the function is accumulating work and consequently the maximum queue-size is growing leading to the delay for passing through that function growing as well.

Lemma 1. *With a constant input rate $r_0(t) = r$, along with all functions switching on/off their additional machine with a common period T, the maximum queue size q_i^{max} at function F_i is*

$$q_i^{\mathsf{max}} = T \times \alpha_i, \tag{14}$$

where

$$\begin{aligned}\alpha_i = \max\big\{&\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big),\\ &(1-\rho_{i-1})(\bar{s}_{i-1}\rho_{i-1} - \bar{s}_i\rho_i),\\ &\rho_{i-1}\big(\bar{s}_{i-1}(1-\rho_{i-1}) - \bar{s}_i(1-\rho_i)\big),\\ &(1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1})\big\},\end{aligned}$$

with ρ_i as defined in (9), and T being the period of the switching scheme, common to all functions.

Proof: Due to limited space the proof is shown in a technical report published at Lund University Publications, [23]. □

The expression of q_i^{max} in (14) suggests that the maximum queue-length is always bounded with respect to the input rate to the service-chain, as shown in Corollary 1. The intuition behind this is that regardless the seize of the input rate, it is possible to find a number $\bar{m}_i$ such that $(\bar{m}_i+1)\bar{s}_i > r$, hence one can always mach the input rate.

Corollary 1. *The maximum queue qize q_i^{max} at any function F_i is bounded, regardless of the rate r of the input.*

Proof: From the definition of ρ_i in Eq. (9), it always holds that $\rho_i \in [0,1)$. Hence, from the expression of (14), it follows that q_i^{max} is always bounded. □

Finally, to solve the optimal design problem one has to relate the period T and the end-to-end delay:

Lemma 2. *With a constant input rate, $r_0(t) = r$, the longest end-to-end delay $\hat{D}_{i,n}$ for any request passing through functions F_1 thru F_n is*

$$\hat{D}_{1,n} = T \times \sum_{i=1}^{n} \delta_i. \tag{15}$$

with δ_i being an opportune constant that depends on r, $\bar{s}_i$, and $\bar{s}_{i-1}$ given in Table 1, with the four different cases given in Table 2.

Proof: Due to limited space the proof is shown in a technical report published at Lund University Publications, [23]. □

Case	δ_i
Case (1a)	$\frac{1}{r}\bar{s}_i\rho_i^2\frac{\bar{s}_i(1-\rho_i)-\bar{s}_{i-1}(1-\rho_{i-1})}{\bar{s}_{i-1}(1-\rho_{i-1})+\bar{s}_i\rho_i}$
Case (1b)	0
Case (2a)	$\begin{cases}\frac{1}{r}\big(\bar{s}_{i-1}\rho_{i-1}(\rho_{i-1}-\rho_i)+ \\ \quad+(1-\rho_i)(\bar{s}_i\rho_i-\bar{s}_{i-1}\rho_{i-1})\big), & T_i^{\mathsf{on}}\geq T_{i-1}^{\mathsf{on}} \\ \frac{1}{r}\Big(\rho_i\big(\bar{s}_i(1-\rho_i)-\bar{s}_{i-1}(1-\rho_{i-1})\big)+ \\ \quad+\bar{s}_{i-1}(\rho_{i-1}-1)(\rho_{i-1}-\rho_i)\Big), & T_i^{\mathsf{on}}<T_{i-1}^{\mathsf{on}}\end{cases}$
Case (2b)	$\frac{1}{r}\bar{s}_i(1-\rho_i)^2\frac{\bar{s}_i\rho_i+\bar{s}_{i-1}\rho_{i-1}}{\bar{s}_i(1-\rho_i)+\bar{s}_{i-1}\rho_{i-1}}$

TABLE 1: The opportune constant δ_i given for each of the four cases (presented in Table 2).

Case (1a) Case (1b)	$(\bar{m}_i+1)\bar{s}_i\geq(\bar{m}_{i-1}+1)\bar{s}_{i-1}$ $(\bar{m}_i+1)\bar{s}_i<(\bar{m}_{i-1}+1)\bar{s}_{i-1}$	$\bar{m}_i\bar{s}_i\geq\bar{m}_{i-1}\bar{s}_{i-1}$
Case (2a) Case (2b)	$(\bar{m}_i+1)\bar{s}_i\geq(\bar{m}_{i-1}+1)\bar{s}_{i-1}$ $(\bar{m}_i+1)\bar{s}_i<(\bar{m}_{i-1}+1)\bar{s}_{i-1}$	$\bar{m}_i\bar{s}_i<\bar{m}_{i-1}\bar{s}_{i-1}$

TABLE 2: Table for finding which of the four possible cases the i'th function in the chain belong to. Each case depend on the nominal services speeds of the function and its preceding function.

3.1. Solution to the optimization problem

With the relationships between the cost, the end-to-end delay, and the period established, the optimization problem (6) can be written as

$$J(T)=aT+\sum_{i:T<\overline{T}_i}\jmath_i^{\mathsf{c}}(1-\rho_i)+\sum_{i:T\geq\overline{T}_i}\jmath_i^{\mathsf{c}}\frac{\Delta_i}{T}+J^{\mathsf{lb}},\quad(16)$$

where J^{lb} is the lower bound given by (7), $a=\sum_{i=1}^n\jmath_i^{\mathsf{q}}\alpha_i$, with α_i given by Lemma 1, and $\overline{T}_i$ is the value of the period below which it is not feasible to switch the additional machine off and then on again, given by (12):

$$T<\overline{T}_i\ \Leftrightarrow\ T_i^{\mathsf{off}}<\Delta_i.$$

In fact, $\forall i$ with $T<\overline{T}_i$ we pay the full cost of having $\bar{m}_i+1$ machines always on.

The deadline constraint in (6), can be simply written as

$$T\leq c:=\frac{D^{\mathsf{max}}}{\sum_{i=1}^n\delta_i},$$

with δ_i given in Lemma 2, Table 1.

The cost $J(T)$ of (16) is a continuous function of one variable T. It has to be minimized over the closed interval $[0,c]$. Hence, by the Weierstaß's extreme-value theorem, it has a minimum. To find this minimum, we just check all (finite) points at which the cost is not differentiable and the ones where the derivative is equal to zero. Let us define all points in $[0,c]$ in which $J(T)$ is not differentiable:

$$\mathcal{C}=\{\overline{T}_i:\overline{T}_i<c\}\cup\{0\}\cup\{c\}.\quad(17)$$

We denote by $p=|\mathcal{C}|\leq n+2$ the number of points in $\mathcal{C}$. Also, we denote by $c_k\in\mathcal{C}$ the points in $\mathcal{C}$ and we assume they are ordered increasingly $c_1<c_2<\ldots<c_p$. Since the cost $J(T)$ is differentiable over the open interval (c_k,c_{k+1}), the minimum may also occur at an interior point of (c_k,c_{k+1}) with derivative equal to zero. Let us denote by $\mathcal{C}^*$ the set of all interior points of (c_k,c_{k+1}) with derivative of $J(T)$ equal to zero, that is

$$\mathcal{C}^*=\{c_k^*:k=1,\ldots,p-1,\ c_k<c_k^*<c_{k+1}\}\quad(18)$$

with

$$c_k^*=\sqrt{\frac{\sum_{i:\overline{T}_i<c_{k+1}}\jmath_i^{\mathsf{c}}\Delta_i}{a}}.$$

Then, the optimal period is given by

$$T^*=\arg\min_{T\in\mathcal{C}\cup\mathcal{C}^*}\{J(T)\}.\quad(19)$$

3.2. Feedback for increased robustness

In this section a feedback-law will be derived to increase the robustness and stability of the system. The goal is to enable the system to handle impulse disturbances of mass d_i occurring at the i'th function, i.e. d_i packets suddenly appear at the tail of the i'th queue, as well as to be able to handle a stochastic input. The feedback will use information about deviations from the expected queue-sizes and use this to dynamically change the on-time of the additional machines in order to drive the system back to the desired queue-sizes and, as illustrated in Figure 4 where the nominal schedule is shown in green and the adjusted, extra on-time for the i'th function highlighted in red. By extending the on-time during the second period the extra work introduced by the impulse-disturbance is processed and the system is driven back to the desired state. What is also highlighted in Figure 4 is that one can use an impulse disturbance to model both modeling errors as well as time-varying input. For instance, denoting the expected queue-size by $q_i(t)$ and the true queue-size by by $\tilde{q}_i(t)$ one can model the difference of them at time t^* by an impulse disturbance of mass $d_i=\tilde{q}_i(t^*)-q_i(t^*)$. One cause for such deviations might be a stochastic input rate.

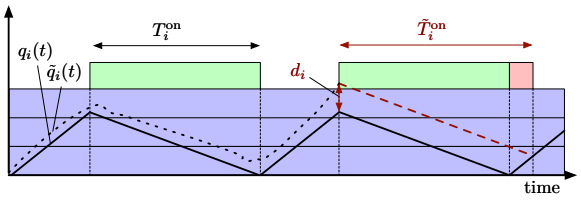

Figure 4: Illustration of the idea to alter the on-time of the additional machine in order to handle the extra load caused by the impulse disturbance. The blue bars symbolize the machines that are always on, and the green symbolize when the extra machine is supposed to be on, and the red bar highlight the additional on-time for the extra machine needed to process the extra work introduced by the impulse disturbance. The solid (–) line show the expected queue-size of the function, and the dashed (- - -) show the true queue-size. One can thus use the impulse disturbance as a tool to model the difference between these when the additional machine starts.

When an impulse disturbance of mass d_i appear at function F_i the nominal on-time T_i^{on} will not be sufficient to process both the residual work, i.e. the work not processed by the $\bar{m}_i$ machines, and the impulse disturbance. Hence, at the end of T_i^{on} there will still be d_i too many packets in the queue. Moreover, without any feedback law to adjust T_i^{on} the extra load will never be processed.

The time needed by the additional machine to process the impulse disturbance is $d_i/\bar{s}_i$, which should thus be added to the nominal on-time T_i^{on}. Denoting the adjusted on-time by $\tilde{T}_i^{\mathsf{on}}$ it should thus be given by

$$\tilde{T}_i^{\mathsf{on}} = T_i^{\mathsf{on}} + \frac{d_i}{\bar{s}_i}. \tag{20}$$

However, since the on-time can only be extended by converting off-time into on-time, it might very well be that $d_i/\bar{s}_i > T_i^{\mathsf{off}}$, implying that there is not sufficient off-time in the next period to convert to on-time in order to process the extra work caused by the impulse disturbance. In fact, assuming that no additional disturbances occur during the processing of d_i, the function will need $\lceil (d_i/\bar{s}_i)/T_i^{\mathsf{off}} \rceil$ periods before the extra work is fully processed. Hence, the total time needed is

$$\tilde{T}_i^{\mathsf{on}} = T \underbrace{\left\lfloor \frac{d_i/\bar{s}_i}{T_i^{\mathsf{off}}} \right\rfloor}_{\text{number of full periods needed}} + T_i^{\mathsf{on}} + T_i^{\mathsf{off}} \overbrace{\left(\frac{d_i/\bar{s}_i}{T_i^{\mathsf{off}}} - \left\lfloor \frac{d_i/\bar{s}_i}{T_i^{\mathsf{off}}} \right\rfloor \right)}^{\text{fraction of final } T_i^{\mathsf{off}} \text{ needed} \in [0,1)}.$$

Here one should note that $\tilde{T}_i^{\mathsf{on}} \to \infty$ as $T_i^{\mathsf{off}} \to 0$, therefore, should $\tilde{T}_i^{\mathsf{on}}$ grow very large it might be favorable to switch on yet another machine. If such a thing would happen it would thus need to switch between using $\bar{m}_i + 1$ and $\bar{m}_i + 2$ machines, which is the problem studied in this paper.

To allow for this impulse disturbance to be accepted by the buffer there must be space in the buffer. Therefore, one would have to increase the maximum queue-size $q_i^{\max}$ given by Lemma 1 to

$$\tilde{q}_i^{\max} = q_i^{\max} + d_i.$$

Naturally one might wonder if this affects the solution of the optimization problem (6) and the answer is no. In the cost function (16), the added queue-size would add a cost

$$\sum_{i=1}^{n} j_i^{\mathsf{q}} d_i$$

which does not depend on the optimization variable T, hence implying a linear shift of the cost and that the same optimal period T still holds. The same holds for the additional time needed to process the impulse disturbance, implying a larger computation cost.

3.3. Designing the schedule

To implement this in the real system, one would have to know when to start the extra machines, i.e. to derive a schedule. With the period T by which the additional machines should be switched on/off, along with the adaptive on-time $\tilde{T}_i^{\mathsf{on}}$ to handle impulse disturbances and variations in the input, the only thing one needs to know before designing a schedule is *when to start the additional machine for the first time*.

With the schedule by being periodic with a period T, the the $(k+1)$'th time the additional machine in the i'th function starts is given by

$$t_{i,k}^{\mathsf{on}} = t_{i,0}^{\mathsf{on}} + kT, \quad i = 1, 2, \ldots, \quad \forall k \geq 1$$

where $t_{i,0}^{\mathsf{on}} \geq 0$ is the first time the additional machine starts in the function. Similarly, the $(k+1)$'th time the additional machine should stop is given by

$$t_{i,k}^{\mathsf{off}} = t_{i,k}^{\mathsf{on}} + \tilde{T}_i^{\mathsf{on}}, \quad i = 1, 2, \ldots, \quad \forall k \geq 1$$

where $\tilde{T}_i^{\mathsf{on}}$ is given by the feedback-law (20), but with d_i being the difference between the expected queue size at this time-instance and the actual queue-size, as shown later. Hence, it remains to define $t_{i,0}^{\mathsf{on}}$ and d_i.

When proving Lemma 1 and 2 in [23] a by-product was the optimal time to start the additional machine as well as the expected queue-size at that time. It was shown that $t_{i,0}^{\mathsf{on}}$ can be expressed relative to $t_{i-1,0}^{\mathsf{on}}$ and that this relationship, as well as the expected queue-size $q_i(t_{i,0}^{\mathsf{on}}) = q_i^{\mathsf{on}}$, depend on $\bar{s}_i$ and $\bar{s}_{i-1}$. Moreover, this dependency can be expressed by the four different cases given in Table 2. Each case giving a different expression for $t_{i,0}^{\mathsf{on}}$ and q_i^{on} as shown in Table 3. Here it should be noted that for the first function in the chain, one can regard the input r as a "dummy function" F_0, preceding F_1, with $\bar{s}_0 = r$, $\bar{m}_0 = 1$, and $\rho_0 = 0$, leading to the first function belonging to Case (2b). Moreover, for this "dummy function", $t_{0,0}^{\mathsf{on}}$ is assumed to be 0. Finally, the on-time computed when the additional machine starts every period, is given by

$$\tilde{T}_{i,k}^{\mathsf{on}} = \min\Big(T,\, T_i^{\mathsf{on}} + \frac{q_i^{\mathsf{on}} - q_i(t_{i,k}^{\mathsf{on}})}{\bar{s}_i}\Big),$$

where the $\min()$ ensures that if there is a large difference between the expected actual queue-sizes the additional machine is kept on for a period, and then a new on-time is computed. In Figure 5 one can see an example-schedule, without disturbances, for a function F_i belonging to Case (1a).

Case (1a)	$t_{i,0}^{\mathsf{on}} = t_{i-1,0}^{\mathsf{on}} + T\rho_i \frac{\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})}{\bar{s}_{i-1}(1-\rho_{i-1}) + \bar{s}_i\rho_i}$ $q_i^{\mathsf{on}} = T\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big)$
Case (1b)	$t_{i,0}^{\mathsf{on}} = t_{i-1,0}^{\mathsf{on}}$ $q_i^{\mathsf{on}} = 0$
Case (2a)	$t_{i,0}^{\mathsf{on}} = t_{i-1,0}^{\mathsf{on}} + T(\rho_{i-1} - \rho_i)$ $q_i^{\mathsf{on}} = \begin{cases} T(1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1}), & T_i^{\mathsf{on}} \geq T_{i-1}^{\mathsf{on}} \\ T\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big), & T_i^{\mathsf{on}} < T_{i-1}^{\mathsf{on}} \end{cases}$
Case (2b)	$t_{i,0}^{\mathsf{on}} = t_{i-1,0}^{\mathsf{on}} + T(1-\rho_i)$ $q_i^{\mathsf{on}} = T(1-\rho_i)(\bar{s}_i\rho_i + \bar{s}_{i-1}\rho_{i-1})$

TABLE 3: Table showing the start-time, $t_{i,0}^{\mathsf{on}}$, for the additional machine and the expected queue-size, $q_i^{\mathsf{on}} = q_i(t_{i,0}^{\mathsf{on}})$, of the i'th function depending on which of the four cases the function belong to.

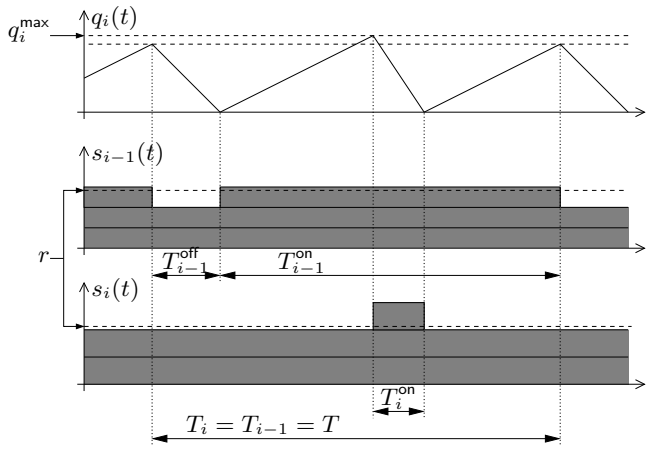

Figure 5: Case (1a): service schedule and queue $q_i(t)$. In this example: $r = 17$, $\bar{s}_{i-1} = 6$, $\bar{s}_i = 8$, $T = 120$, $T_{i-1}^{\text{on}} = 100$, $T_i^{\text{on}} = 15$, $q_i^{\max} = 90$.

4. Example

In this section we investigate the analysis of this paper using an example with two functions and an input rate of $r = 17 \times 10^3$ packets per second. Every request has an end-to-end-deadline of $D^{\max} = 0.02$ seconds. The parameters of the two functions are reported in Table 4.

i	$\bar{s}_i$ (pps)	$\jmath_i^{\text{c}}$	$\jmath_i^{\text{q}}$	Δ_i (s)
1	6×10^3	6	0.5×10^{-3}	0.01
2	8×10^3	8	0.5×10^{-3}	0.01

TABLE 4: Parameters of the example.

As mentioned earlier, the input $r(t) = r$ can be seen as dummy function F_0 preceding F_1, with $\bar{s}_0 = r$, $\bar{m}_0 = 1$, and $\rho_0 = 0$. When deriving the schedule, it follows from (8) and (9) that $\bar{m}_1 = \bar{m}_2 = 2$, and $\rho_1 = \frac{5}{6}$, $\rho_2 = \frac{1}{8}$, implying that both functions must always keep two machines on, and then periodically switch a third one on/off. This leads to $\overline{T}_1 = 60.0 \times 10^{-3}$ and $\overline{T}_2 = 11.4 \times 10^{-3}$, where $\overline{T}_i$ is the threshold period for function F_i, as defined in (12).

From Lemma 1 it follows that the parameter a of the cost function (16) is $a = 0.792$, while from Lemma 2 the parameters δ_i determining the queuing delay introduced by each function, are $\delta_1 = 49.0 \times 10^{-3}$ and $\delta_2 = 22.1 \times 10^{-3}$, which in turn leads to

$$c = \frac{D^{\max}}{\delta_1 + \delta_2} = \frac{0.02}{71.1 \times 10^{-3}} = 281 \times 10^{-3}.$$

Since $\overline{T}_2 < \overline{T}_1 < c$, the set $\mathcal{C}$ of (17) containing the boundary is

$$\mathcal{C} = \{0, \underbrace{0.00114}_{\overline{T}_2}, \underbrace{0.060}_{\overline{T}_1}, \underbrace{0.281}_{c}\}.$$

To compute the set $\mathcal{C}^*$ of interior points with derivative equal to zero defined in (18), which is needed to compute the period with minimum cost from (19), we must check all intervals with boundaries at two consecutive points in $\mathcal{C}$. In the interval $(0, \overline{T}_2)$ the derivative of J is never zero. When checking the interval $(\overline{T}_2, \overline{T}_1)$, the derivative is zero at

$$c_1^* = \sqrt{\frac{\jmath_2^{\text{c}} \Delta_2}{a}} = 0.318,$$

which, however, falls outside the interval. Finally, when checking the interval $(\overline{T}_1, c)$ the derivative is zero at

$$c_2^* = \sqrt{\frac{\jmath_1^{\text{c}} \Delta_1 + \jmath_2^{\text{c}} \Delta_2}{a}} = 0.421 > c = 0.281.$$

Hence, the set of points with derivative equal to zero is $\mathcal{C}^* = \emptyset$. By inspecting the cost at points in $\mathcal{C}$ we find that the minimum occurs at $T^* = c = 0.281$, with cost $J(T^*) = 34.7$. The period of T^* results in a schedule with a state-space trajectory for the two queues shown in Figure 6.

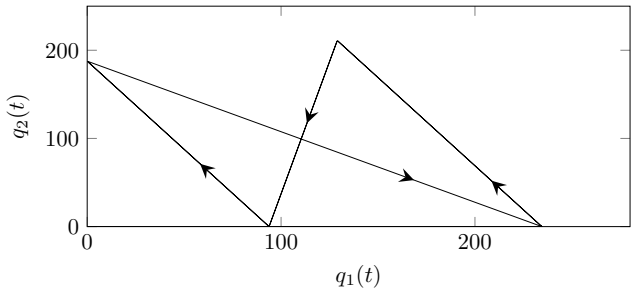

Figure 6: Nominal state-space trajectory of the queues for the two functions in the example. One can see that, when the system has a constant input, and no disturbance, the trajectory follows a limit-cycle.

4.1. Impulse disturbance

Should the system fall victim to an impulse disturbance, i.e. that d_i packets suddenly appear at the tail of the queue in the i'th function, the nominal limit-cycle of the queues (shown in Figure 6) will be perturbed. This is illustrated in Figure 7 where we gave the first function an impulse disturbance of mass d_1 after 1.5 periods. With no feedback-law in place, the nominal limit-cycle is never recovered. Instead it will be perturbed a distance d_i in the q_i'th direction of the state-space diagram.

When extending the system with the feedback-law described in Section 3.2 the functions will dynamically change the on-time of the additional machines. This leads to the nominal limit-cycle being recovered after the initial impulse disturbance, as shown in Figure 8. One can see that the recovery takes around two periods, ending when the blue trajectory hits the black one.

4.2. Stochastic input

A natural question is whether the feedback-law can handle a stochastic input, and not just the occasional impulse disturbance. We denote the nominal input rate $r_0 = 17 \times 10^3$, and then extend the input to the system to be stochastic, with a uniform distribution from the interval $r(t) \in [0.9,\ 1.1] \times r_0$.

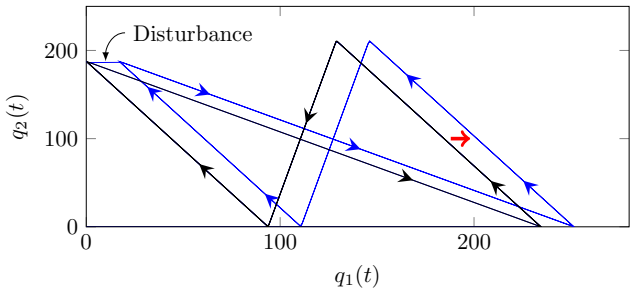

Figure 7: When the system is given an impulse disturbance of mass d_1 to the first function, the nominal limit-cycle (black) is perturbed to the right, giving a new limit-cycle (blue). Without a feedback-law, the nominal limit-cycle is never recovered.

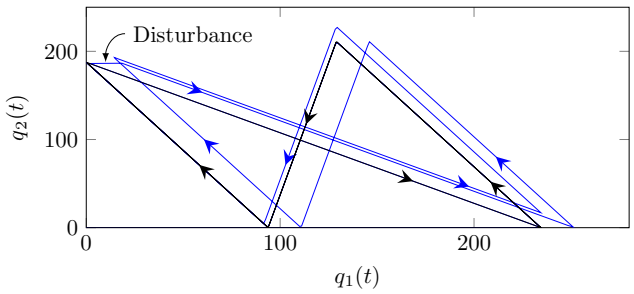

Figure 8: Using the feedback-law it successfully recovers the nominal limit-cycle (black) an impulse disturbance of mass d_1. The blue trajectory shows the recovery of the nominal limit-cycle, which takes about two periods.

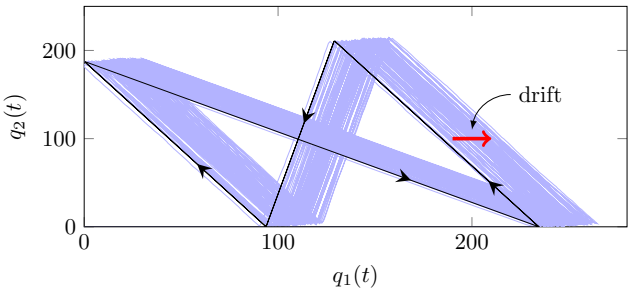

Figure 9: State-space trajectory (blue) for a system with a stochastic input but with no feedback from the queue-size. One can see the the trajectory drifts away from the nominal limit-cycle (black).

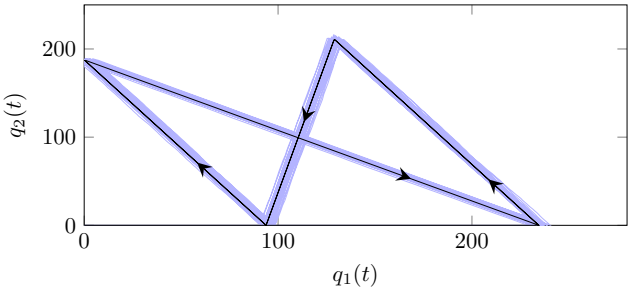

Figure 10: State-space trajectory (blue) for a system with a stochastic input where feedback from the queue-size pulls the system back around the nominal limit-cycle (black).

Should this input be used for the original system, without feedback, the nominal limit-cycle would drift every time the input rate is larger than the nominal one, i.e. when $r(t) > r_0$. This can be seen in Figure 9 where system was run for 1000 periods. The black trajectory show the nominal limit-cycle, and the blue show the trajectory of the simulation. One can see that the system does not converge back to the nominal limit-cycle. This implies that the queue-size would grow with t, $q_i(t) \to \infty$, as $t \to \infty$, making it impossible to dimension the necessary buffer size.

With the feedback-law, however, the system will ensure that the nominal limit-cycle is restored by dynamically changing the on-time for each additional function every time it is turned on, hence making it possible to dimension the queue-sized. This is illustrated in Figure 10 showing the the state-space trajectory of the system is centered around the nominal limit-cycle during a simulation length of a 1000 periods.

5. Summary

In this paper we have developed a general mathematical model for a service-chain residing in a cloud environment. This model includes an input model, a service model, and a cost model. The input-model defines the input-stream of requests to each NFV along with end-to-end deadlines for the requests, meaning that they have to pass through the service-chain before this deadline. In the service-model, we define an abstract model of a NFV, in which requests are processed by a number of machines inside the service function. It is assumed that each function can change the number of machines that are up and running, but doing so is assumed to take some time. The cost-model defines the cost for allocating compute- and storage capacity, and naturally leads to the optimization problem of how to allocate the resources.

The optimization problem for controlling the resources of the network functions is analyzed and solved under the assumption of a constant input-stream of requests as well as having every function in the chain switch on/off their extra machine with the same period, although not necessarily having the them on for the same duration. The solution derived with these assumptions is then augmented with a feedback-law, allowing the machines to dynamically adjust the necessary on-time for the extra machine depending on whether the queue-size match the expected queue-size, or whether they deviate. This leads to the system being able to handle both impulse disturbances as well as a stochastic input.

Acknowledgements. The authors would like to thank Karl-Erik Årzén and Bengt Lindoff for the useful comments on early versions of this paper.

References

[1] ETSI, "Network Functions Virtualization (NFV)," https://portal.etsi.org/nfv/nfv_white_paper.pdf, October 2012.

[2] ——, "Network Functions Virtualization (NFV); Use Cases," October 2013.

[3] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 69–84.

[4] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable low latency for data center applications," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 9:1–9:14. [Online]. Available: http://doi.acm.org/10.1145/2391229.2391238

[5] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU resource management for real-time cloud computing," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 179–186.

[6] K. W. Tindell, A. Burns, and A. Wellings, "An extendible approach for analysing fixed priority hard real-time tasks," *Journal of Real Time Systems*, vol. 6, no. 2, pp. 133–152, Mar. 1994.

[7] J. Palencia and M. G. Harbour, "Offset-based response time analysis of distributed systems scheduled under EDF," in *15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003.

[8] R. Pellizzoni and G. Lipari, "Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling," *Journal of Computer and System Sciences*, vol. 73, no. 2, pp. 186–206, Mar. 2007.

[9] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings of the 15-th IEEE Real-Time Systems Symposium*, Dec. 1994, pp. 215–227.

[10] S. 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*, 2006, pp. 191–198.

[11] N. Serreli, G. Lipari, and E. Bini, "Deadline assignment for component-based analysis of real-time transactions," in *2nd Workshop on Compositional Real-Time Systems*, Washington, DC, USA, Dec. 2009.

[12] ——, "The demand bound function interface of distributed sporadic pipelines of tasks scheduled by EDF," in *Proceedings of the 22-nd Euromicro Conference on Real-Time Systems*, Bruxelles, Belgium, Jul. 2010.

[13] S. Hong, T. Chantem, and X. S. Hu, "Local-deadline assignment for distributed real-time systems," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1983–1997, Jul. 2015.

[14] A. Rahni, E. Grolleau, and M. Richard, "Feasibility analysis of non-concrete real-time transactions with edf assignment priority," in *Proceedings of the 16-th conference on Real-Time and Network Systems*, Rennes, France, Oct. 2008, pp. 109–117.

[15] L. Kleinrock, *Queueing Systems*. John Wiley & Sons, 1975.

[16] D. Henriksson, Y. Lu, and T. Abdelzaher, "Improved prediction for web server delay control," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Jun. 2004, pp. 61–68.

[17] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity*. Wiley New York, 1992, vol. 3.

[18] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.

[19] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[20] J.-Y. Le Boudec and P. Thiran, *Network Calculus: a theory of deterministic queuing systems for the internet*, ser. Lecture Notes in Computer Science. Springer, 2001, vol. 2050.

[21] S. Chakraborty and L. Thiele, "A new task model for streaming applications and its schedulability analysis," in *Design, Automation and Test in Europe Conference and Exposition*, Mar. 2005, pp. 486–491.

[22] R. Bonafiglia, I. Cerrato, F. Ciaccia, M. Nemirovsky, and F. Risso, "Assessing the performance of virtualization technologies for nfv: a preliminary benchmarking," in *2015 Fourth European Workshop on Software Defined Networks*. IEEE, 2015, pp. 67–72.

[23] V. Millnert, J. Eker, and E. Bini, "Cost minimization of network services with buffer and end-to-end deadline constraints," p. 11, 09 2016. [Online]. Available: https://lup.lub.lu.se/search/publication/8c7b837e-bca3-4375-bb9d-28ce6bbc889a