Achieving predictable and low end-to-end latency for a cloud-robotics network

Millnert, Victor; Eker, Johan; Bini, Enrico

2018

Document Version:
Publisher's PDF, also known as Version of record

Achieving predictable and low end-to-end latency for a cloud-robotics network

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

Abstract

To remain competitive in the field of manufacturing today, companies must make their industrial robots smarter and allow them to collaborate with one another in a more effective manner. This is typically done by adding some form of learning, or artificial intelligence (AI), to the robots. These learning algorithms are often packaged as cloud functions in a remote computational center since the amount of computational power they require is unfeasible to have at the same physical location as the robots.

Using and augmenting the robots with these such cloud functions has usually not been possible since the robots require a very low and predictable end-to-end latency–something which is difficult to achieve when involving cloud functions. Moreover, different sets of robots will have different end-to-end latency requirement, despite using the same network of cloud functions. However, with the introduction of 5G and network function virtualization (NFV) it does become possible. With this technology it becomes possible to control the amount of resources allocated to the different cloud functions and thereby gives us control over the end-to-end latency. By controlling this in a smart way it will become possible to achieve a very low and predictable end-to-end latency.

In this work we address this challenge by deriving a rigorous mathematical framework that models a general network of cloud functions. On top of this network several applications are hosted. Using this framework we propose a generalized AutoSAC (automatic service- and admission controller) that builds on previous work by the authors [1], [2]. In the previous work the system was only capable of handling a single set of cloud functions, with a single application hosted on top of it. With the contributions of this paper it becomes possible to host multiple applications on top of a larger, general network of cloud functions. It also allows for each application to have its own end-to-end deadline requirement.

The contributions of this paper can be summed up by the following four parts:

a) *Input prediction:* To achieve a good prediction of propose a communication scheme between the cloud functions. This allows a for a quicker reaction to changes of the traffic rates and in the end a better utilization of the resources allocated to the cloud functions.

b) *Service control:* With a small theorem we are able to show a simplification of the control law derived in the previous work. This can be especially useful when controlling cloud functions that make use of a large number of virtual machines or containers.

c) *Admission control:* To be able to ensure that the end-to-end latency is low and predictable we equip every cloud function with an intermediary node deadline. To enforce the node deadlines we propose a novel admission controller capable of achieving the highest possible throughput while still guaranteeing that every packet that is admitted will meet the node deadline. Furthermore, we show that the computation necessary for this can be done in constant time, implying that it is possible to enforce a time-varying node deadline.

d) *Selection of node deadlines:* The problem of assigning intermediary node deadlines in a way that enforce the global end-to-end deadlines is addressed by investigating how different node deadlines affect the performance of the network. The insights from this then used to set up a convex optimization problem for the assignment problem.

CONTENTS

I. Introduction

To remain competitive in the field of manufacturing today, companies must make their industrial robots smarter and allow them to collaborate with one another in a more effective manner. This is typically done by adding some form of learning, or artificial intelligence (AI), to the robots. Naturally, these learning algorithms require a substantial amount of computational power, especially when there are multiple industrial-robots collaborating and learning tasks together. The amount of computational power necessary is often unfeasible to have at the same physical location as the robots themselves. Therefore, they have to be connected to a remote computational center, such as a local data-center, an edge data-center, or even a remote data-center. In such a remote computation center, the learning/AI algorithms will often be packaged as a *cloud functions*. This allows many robots to simultaneously use the same cloud functions since it is possible to dynamically scale the resources allocated to them. Moreover, it also allows a single robot to use many different functions, often in a chained manner (an example of which could be a chained stream-processing AI-framework such as a classification neural-net).

While this has the potential to substantially change how manufacturing is done, the other side of the coin is that *collaborating industrial robots require a predictable- and low end-to-end latency, often in the order of milliseconds*. Setting up such a network of collaborating industrial robots is very tedious, even without the smart cloud functions. Add cloud functions to the network and it becomes very difficult. Setting up many different networks that use cloud functions is typically not an option today.

With the coming technology promised by 5G and network function virtualization (NFV) it will be possible to build and easily set up a network with the predictable and low end-to-end required by smart manufacturing. However, 5G and NFV is "only" the enabling technology for this and one will only achieve a low and predictable end-to-end latency if one control the network in a "correct way".

In Figure 1 a network of collaborating robots using cloud functions is illustrated, with one set of robots (blue circles) are collaborating and sending data through a set of cloud functions (green circles) to another set or robots (red circles). It highlights the fact that controlling the capacity of each of the cloud functions in order to achieve a predictable and low end-to-end latency is indeed a complex task, especially since the complexity of the network grows fast with the number of collaborating robots and the number of cloud functions used.

In this work we address the challenge of controlling a network of cloud functions in a way to that achieves a predictable and low end-to-end latency. The use-case for such a system, as mentioned above, is for smart manufacturing where many industrial robots collaborate, using remote cloud functions, to solve a problem together. We propose a generalized AutoSAC (automatic service- and admission control), that builds on previous work [1], [2]. In the early works, however, the system was only capable of handling a

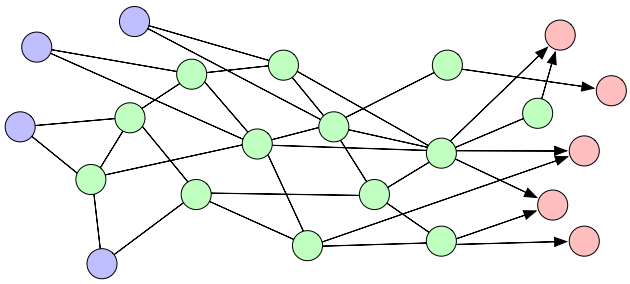


Fig. 1: A simple illustration of a network of industrial robots (blue circles) that are using a set of different cloud functions (green circles) to process signals and data which in turn are sent to the collaborating robots (red circles). The illustration gives an idea that the task of scaling the resources allocated to the cloud functions is very complex, especially when, despite the interaction of multiple different packet flows, one wish to achieve a predictable and low end-to-end latency.

single chain of cloud functions, a single packet-flow, and a single end-to-end deadline. The work presented in this paper is therefore a generalization necessary to handle the new network of cloud functions with multiple packet-flows, and multiple end-to-end deadlines defined on top of it.

Finally, something enabled, but not addressed, by this work is the possibility of allowing packet-flows to come and go. This could for instance be when a new set of collaborating robots which to start working, or perhaps leave the network. This work enable this through the proposition of a new admission controller which is able to enforce a dynamic deadline for a cloud function. The computation required by the admission controller to do this can be done in constant time. Allowing dynamic deadlines for the different functions in the network will allow the end-to-end deadlines to change. In the end, this would allow the network to move between different deadline configurations, effectively allowing new packet-flows, with new end-to-end deadlines, to join the network as well as present packet-flows to leave the network.

Related works

There has been a number works written on the topic of controlling resources in the cloud and the area of network function virtualization. The majority of them focus on orchestration, i.e. the problem of deciding where in the physical world the virtual resources should be allocated. A few works differ, however, in the way that they instead consider the problem of controlling the NFV graphs with respect to some end-to-end goals. For instance Lin et al. [3] do a static one-time orchestration with the right amount of resources to satisfy some end-to-end requests. Shen et al. [4] develop a management framework, vConductor, for realizing end-to-end virtual network services. However, they are not considering timing-sensitive applications with deadlines for the packets moving through the chain, which is done by Li et al [5] where they present a design and implementation of NFV-RT that aims at controlling NFVs with soft Real-Time guarantees, allowing packets to have deadlines

Despite the dynamic nature of the traffic, the NFV graphs will encounter only a few works consider it and aim at designing an elastic, dynamic resource controller to counter the problem. In [6] Mao et al. develop a mechanism for auto-scaling VNF resources to meet user-specified performance goal. Another work that addresses the problem of meeting performance goals despite the dynamic traffic is [7]. They achieve it by doing load-balancing with a SDN controller between the VNFs. A great work also combining flow scheduling and resource allocation is [8] where they develop a neat mathematical model used as foundation for their synthesis. Other works focusing on developing a model of a VNF is [9], and [10].

The classic method to guarantee end-to-end deadlines of transaction is by holistic analysis [11], in which schedulability analysis at each node is iterated until the convergence of the response times of each transaction is reached. Pellizzoni and Lipari [12] improved the holystic analysis by using offset rather than jitter of tasks. Lorente et al. [13] extended the holysic analysis to the case with nodes running at a fraction of computing capacity (abstracted by a bounded-delay time partition with bandwidth and delay). Similarly, Ashjaei et al. [14] proposed resource reservation over each node along the path.

The enforcement of an end-to-end deadline for a sequence of jobs is however addressed by several works, possibly under different terminologies. Di Natale and Stankovic[15] propose to split the E2E deadline proportionally to the local computation time or to divide equally the slack time. Later, Jiang[16] 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.

II. Model and problem formulation

The goal of this section is to derive a mathematical framework for modeling the network of cloud functions or virtual network functions (VNFs) such as the one described in Section I and illustrated in Figure 1. The framework will be able to model the different packet flows going through the network and allow each of them to have an end-to-end deadline. A simple, abstract version of such a network model is shown in Figure 2. It shows three VNF nodes, each consisting of a number of virtual machines (VMs) deployed on some underlying infrastructure, such as a data center. On top of this are two packet flows,

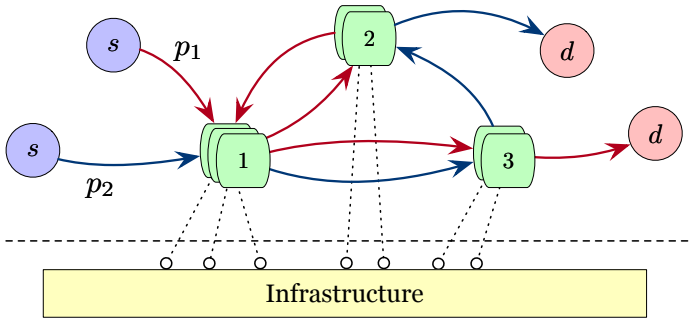


Fig. 2: A simple network with two sources, two destinations, three virtual network nodes (VNFs), and two packet flows p_1 (in red) and p_2 (in blue). Each of the VNF nodes consists of a number of virtual resources, e.g., virtual machines or containers, which are deployed on some commodity hardware infrastructure. One should node that the first packet flow, p_1 traverse the network along a path $\{1, 2, 1, 3\}$ implying that it visits the first VNF node twice, something that will be useful when modeling the typical control-loops described in the Introduction.

depicted in red and blue. One should note that the red packet flow traverse the first node twice, something that is important when deriving a general framework for modeling.

In Section II-A a formal model for the network structure and the packet flows will be derived, followed by Section II-B where the functions of the VNF nodes will be captured and modeled, ending with Section II-C where the problem formulation of this paper is presented.

A. Network and packet flows

To model the network of cloud functions or VNFs we start by describing the connectivity among them as a *directed graph* $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where

- $\mathcal{V}$ is the set of $n = |\mathcal{V}|$ VNF nodes. For convenience, we label the nodes with the integers from 1 to n, that is $\mathcal{V} = \{1, \dots, n\}$;
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges between these nodes. If $(i, i') \in \mathcal{E}$ then an edge from node $i \in \mathcal{V}$ to node $i' \in \mathcal{V}$ exists.

The network graph $\mathcal{G}$ will see a set of f different *flows* traversing the network. Each flow has an *end-to-end deadline* and for the j-th flow this is given by $\mathcal{D}_j$. Moreover, the packets belonging to the j-th flow will visit a specific set of nodes (where they will be processed) in a specific order. This is modeled by the sequence $p_j : \{1, \dots, \ell_j\} \to \mathcal{V}$, with $\ell_j \geq 1$ being the length of the path, such that

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

The function p_j is therefore a mapping from the integers $1, \dots, \ell_j$ to the set of nodes given by $\mathcal{V}$. Hence, $p_j(k)$ is the k-th node of the j-th path. Naturally, equation (1) enforce the existence of an edge between two consecutive nodes in a path of a packet flow. One should note that this model allow for packet flows to traverse a node more than once thus allowing us to model the typical scenario of collaborating robots mentioned in Section I.

The example network illustrated in Figure 2 would, using the framework above, be modeled by a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $\mathcal{V} = \{1, 2, 3\}$ and $\mathcal{E} = \{(1, 2), (2, 1), (1, 3), (3, 1)\}$. Moreover, the two packet flows would be given by $p_1 = \{1, 2, 1, 3\}$ and $p_2 = \{1, 3, 2\}$, and their E2E deadline given by $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively.

B. Node model

In order to derive a good way of controlling the resources allocated to the different cloud functions or VNFs in the network, a good model is required. Such a model will be presented here, and will be used later in Section III to derive the different controllers. In Figure 3 a conceptual version the model and the

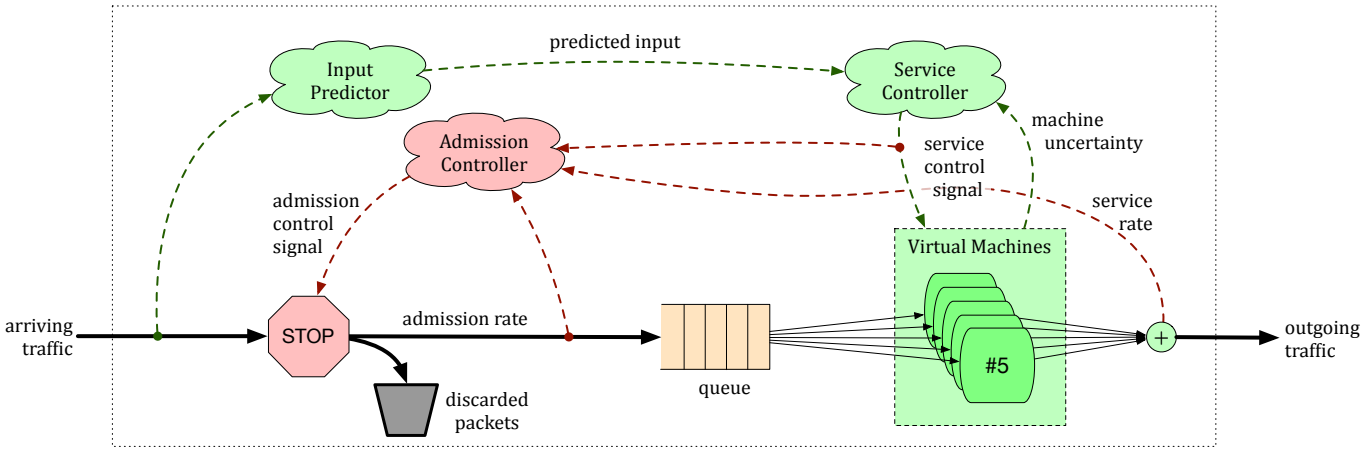


Fig. 3: An overview of the entities of the virtual network functions that is captured in the model developed in this paper. It is assumed that it is possible to measure incoming traffic load and then either admit the packets into a queue, or if there is a possibility of the incoming packets to miss its local deadline, i.e., the deadline for passing through the VNF node, to instead discard the packet. There are a number of instances, i.e., virtual machines, processing the packets in the queue and forwarding them to the next VNF node in the forwarding path.

controllers is illustrated. While the inner workings of the admission controller, input predictor, and service controller will be presented in Section III, the focus during the remainder of this subsection will be on the following parts:

- *traffic flow* – the arrival rate, admission rate, and service rate of the node;
- *machine model* – a processing model of the virtual machines, including the inherent uncertainty of their performance;
- *node latency* – the node latency, i.e., time it takes a packet to pass through the node;
- *node deadline* – the node deadline, i.e., a deadline for a packet to pass through the node.

Traffic flow. At time t the i-th node in the network will see an traffic arriving at a rate of $r_i(t) \in \mathbb{R}^+$ packets per second (pps). The arriving packets are either discarded or admitted into the queue of the node at an *admission rate* of $a_i(t) \in [0,\, r_i(t)]$ pps. At time t the node will have a *queue size* of $q_i(t)$ packets. In order to process the packets in the queue, a number of $m_i(t) \in \mathbb{N}$ virtual machines provide the node with a *maximum processing capacity* of $s_i^{\text{cap}}(t)$ packets per second. Naturally, the node might not always be able to process at maximum capacity, i.e., the queue might be empty, so the actual rate by which the node is processing packets with is given by the *service rate* $s_i(t)$:

$$s_i(t) = \begin{cases} s_i^{\text{cap}}(t) & \text{if } q_i(t) > 0, \\ \min(s_i^{\text{cap}}(t),\, a_i(t)) & \text{else.} \end{cases} \tag{2}$$

As will be shown later, when deriving the node latency and the admission controller it is very convenient to define the *arrival curve* $R_i(t)$, *admission curve* $A_i(t)$, and *service curve* $S_i(t)$ as the cumulative versions of their corresponding lower-cased variables:

$$R_i(t) = \int_0^t r_i(x)\mathrm{d}x, \quad A_i(t) = \int_0^t a_i(x)\mathrm{d}x, \quad S_i(t) = \int_0^t s_i(x)\mathrm{d}x. \tag{3}$$

Machine model. In order to adapt the maximum processing capacity to changes of the arrival rate, it is possible to change and control the number of running virtual machines through the *control signal* $m_i^{\text{ref}}(t) \in \mathbb{N}$. However, since it takes some time to start and stop a virtual machine the number of VMs that are running is a delayed version of this signal, i.e.,

$$m_i(t) = m_i^{\text{ref}}(t - \Delta_i), \tag{4}$$

where $\Delta_i \in \mathbb{R}^+$ is the *time-delay* needed to start and stop a virtual machine.

Every machine instance running in the node has an expected service capacity of $\bar{s}_i$ packets per second, meaning that without the presence of performance uncertainty the node would be able to process packets at a rate of $m_i(t) \cdot \bar{s}_i$. In reality, however, the expected performance rarely correspond to the actual performance. Instead the performance of a VM depend heavily on where it is deployed as well as on what other processes running on the physical server that the VM is hosted on, i.e. on so called resource hogs, [17].

To be able to capture this we introduce a *machine uncertainty* where the machine uncertainty for the k-th VM-instance is given by

$$\xi_{i,k}(t) \in [\xi_i^{\text{lb}}, \xi_i^{\text{ub}}] \text{ pps}, \quad -\bar{s}_i < \xi_i^{\text{lb}} \leq \xi_i^{\text{ub}} < \infty,$$

where ξ_i^{lb} and ξ_i^{ub} are the lower and upper bounds of the machine uncertainty, assumed to be known, for instance through benchmarking. This machine uncertainty is assumed to be fairly constant during the lifetime of an instance but to change when new instances are started or stopped.

The maximum processing capacity of the node can therefore now be expressed as

$$s_i^{\text{cap}}(t) = m_i(t) \cdot (\bar{s}_i + \hat{\xi}_i(t)), \tag{5}$$

where $\hat{\xi}_i(t)$ is the *average machine uncertainty*:

$$\hat{\xi}_i(t) = \frac{1}{m_i(t)} \sum_{k=1}^{m_i(t)} \xi_{i,k}(t). \tag{6}$$

Node latency. Since the time it takes a packet to pass through the set of nodes along its path, e.g., its end-to-end latency, it is very useful to be able to describe the time it takes the packet to pass through the node. This is given by the *node latency* $L_i(t)$:

$$L_i(t) = \inf\{\tau \geq 0 \, : \, A_i(t-\tau) \leq S_i(t)\}. \tag{7}$$

Similarly, and something that will be used in the admission controller derived in Section III-C, is the *upper bound on the expected node latency*:

$$L_i^{\text{ub}}(t) = \inf\Big\{\tau \geq 0 : A_i(t) \leq S_i(t) + \int_t^{t+\tau} m_i(x) \cdot (\bar{s}_i + \xi_i^{\text{lb}}) \mathrm{d}x\Big\}. \tag{8}$$

In plain words this it the expected time it will take a new packet (if admitted into the node) to pass through the node.

The careful reader would notice that to compute $L_i^{\text{ub}}(t)$ one requires information about $m_i(t)$ for the future. However, due to the time-delay of Δ_i time-units needed to start/stop a VM there is actually information about $m_i(x)$ available for the interval $x \in [0, \, t + \Delta_i]$. Since $m_i^{\text{ref}}(t) = m_i(t + \Delta_i)$ and $m_i^{\text{ref}}(x)$ is known for $x \in [0, \, t]$ it is therefore possible to compute the upper bound on the expected delay as long as it is shorter than the time-overhead, something that is sufficient for the admission policy developed in Section III-C.

Node deadline. In the example given in Section I it is assumed that there are many different paths going through the different nodes of the network. With such scenarios in mind, it is convenient to reduce the complexity at the node-level. Therefore, we introduce an intermediate *node deadline* $D_i(t)$. This means that any packet that arrives to the i-th node will only be admitted into the node if it is possible to guarantee that the packet will be processed and exit the function within $D_i(t)$ time-units. This must hold, regardless of which path the packet belongs to. In other words, every packet arriving to the i-th node will have the same *node deadline*, even though they might belong to different paths and have different *end-to-end deadlines*. Naturally, this means that when assigning node deadline one will be constrained by

$$\sum_{k=1}^{\ell_j} D_{p_j(k)} \leq \mathcal{D}_j \qquad \forall j \in 1, \ldots, f. \tag{9}$$

C. Problem formulation

As mentioned in the introduction of this paper, the goal is to derive ways of controlling resources allocated to the network of cloud functions, or VNFs, in order to have predictable and low end-to-end latencies. Informally, this goal can be described as trying to ensure that *"the end-to-end deadlines of the different packet-flows are met, while using as little resources and discarding as few packets as possible"*. To achieve this, we have split the goal into a four of sub-problems:

- *input prediction* – how will the arrival rates change, and how can this information be passed on between the nodes of the network?
- *service control* – how many virtual machines must be running, and how does the machine uncertainty affect this?
- *admission control* – how many packets can be admitted into the node while still guaranteeing that they will meet the node deadline?
- *selection of node deadlines* – what is the optimal way to split the set of end-to-end deadlines into intermediate node deadlines?

Since all of the four sub-parts above interact with each other in a complex manner, it is useful to derive a formal metric when evaluating the overall performance of the system, as will be done later in Section IV. We therefore introduce three such metrics:

a) *availability* $U^{\mathsf{a}}(t)$ – is there a high throughput, or are many packets being discarded?
b) *efficiency* $U^{\mathsf{e}}(t)$ – are the nodes efficient, or are they wasting resources?
c) *utility* $U(t)$ – a combination of the availability and the efficiency.

$$\begin{aligned} U^{\mathsf{a}}(t) &= \tfrac{1}{n}\sum_{i\in\mathcal{V}} u_i^{\mathsf{a}}(x), \quad U^{\mathsf{e}}(t) = \tfrac{1}{n}\sum_{i\in\mathcal{V}} u_i^{\mathsf{e}}(x), \\ U(t) &= \tfrac{1}{n}\sum_{i\in\mathcal{V}} u_i^{\mathsf{a}}(x)\cdot u_i^{\mathsf{e}}(t). \end{aligned} \tag{10}$$

with $u_i^{\mathsf{a}}(t)$ and $u_i^{\mathsf{e}}(t)$ given by

$$\begin{aligned} u_i^{\mathsf{a}}(t) &= \begin{cases} s_i(t)/r_i(t) \ \left(= \dfrac{\text{service}}{\text{demand}}\right) & \text{if } d_i(t) \le D_i \\ 0 & \text{if } d_i(t) > D_i \end{cases} \\ u_i^{\mathsf{e}}(t) &= \frac{s_i(t)}{s_i^{\text{cap}}(t)} \ \left(= \frac{\text{service}}{\text{capacity}}\right) \end{aligned} \tag{11}$$

The intuition behind the choice of these metrics is that it is typically easy to achieve a high efficiency or a high availability. One can for instance choose to overallocate resources in a node, something typically seen today, resulting in a high availability but a poor efficiency, or one can instead choose to have a high efficiency but having to discard many packets. Achieving a high utility requires one to have both a high efficiency and a high availability, something that is very difficult.

The goal when deriving the input prediction, service control, admission control, and the selection of node deadlines will therefore be to do this in a way that maximizes the utility.

III. AutoSAC for a network of cloud functions

In this section we will use the mathematical framework presented in Section II to address and propose solutions to the four sub-problems stated in the problem formulation in Section II-C. To give a brief overview of the solution, we highlight the main points of each solutions here, with a more thorough description given in the following subsection. Before diving into those, however, we provide the underlying timing assumptions.

Input prediction. To improve the prediction of future arrival rates we introduce the notion of *internal and external arrival rates*. They are based on whether the incoming traffic to a node comes directly from another node or directly from a source outside the network. Furthermore, to improve the prediction for nodes located deep in the network topology we introduce a way of passing information

about predicted arrival rates between the nodes in the network. This way, *internal traffic* can be predicted in a very accurate way, greatly improving the reaction times of the nodes in the network.

Service control. To have every node running the "right" number of virtual machines, i.e., not too many nor too few, we derive a control-law optimized with respect to the utility metric described in Section II-C. Furthermore, due to the performance uncertainty of the virtual machines the control-law use a feedback-loop with the purpose of compensating for any deviation from the expected performance of the virtual machines.

Admission control. To ensure that the nodes that are admitted into the nodes are guaranteed to meet the node deadlines we derive a new admission controller. This admission controller is capable of guaranteeing this while still discarding as few packets as possible. Furthermore, the check whether an incoming packet will meet the node deadline or not can be computed instantly. Both of the statements above are proved in a theorem.

Selection of node deadlines. In order to solve the problem of assigning node deadlines we thoroughly investigate the relationship between different node deadlines and the amount of packets that are discarded. This results in a relationship which is used to set up a convex optimization problem of assigning node deadlines.

Timing assumptions. It should be noted that the difficulty when addressing the problems above lies in the different time-scales for starting/stopping instances, the end-to-end deadlines, and the rate-of-change of the arrival rates. Furthermore, since this work builds upon the precious work, presented in [2] and [1], the timing assumptions within this paper remain the same as the one for the earlier works. These assumptions are given below, in Table I.

Parameter	timing assumption
long-term trend change of the input	1min – 1h
time-overhead Δ_i	1s – 1min
end-to-end deadline $P^{\max}$	1µs – 100 ms

TABLE I: Timing assumptions for the end-to-end deadline, the rate-of-change of the input traffic, and the time-overhead for starting/stopping instances.

A. Input prediction

In order to obtain the goal of having a high throughput of packets while still using as little resources, i.e., virtual machines, as possible it is paramount that the prediction of future arrival rates are as accurate as possible. The reason is that it takes Δ_i seconds for the i-th node to start/stop a VM. Recall that the number of running VMs is given by $m_i(t) = m_i^{\text{ref}}(t - \Delta_i)$, hence it takes Δ_i seconds to change this. This means that at time t the node has to make a decision about how many VMs it will need at time $t + \Delta_i$. For this decision to be good, i.e., to yield the highest possible utility, a good prediction of the arrival rate at time $t + \Delta_i$ is necessary.

It is complicated to predict the future arrival rates of a node due to the network structure of the cloud functions. The traffic that pass through a node originates from many different sources. Some parts of the traffic has moved through many other nodes within the network, while some parts comes directly from an outside source. Due to this nature, we find it useful to introduce a distinction between the two. Hence, we introduce the notion of *internal traffic* $r_i^{\text{I}}(t)$ and *external traffic* $r_i^{\text{E}}(t)$. By internal traffic we mean traffic that arrives at the node directly from another node in the network, and by external we mean traffic that arrive directly to the node *without* passing through another node in the network. Together, they form the total arrival rate for the node:

$$r_i(t) = r_i^{\text{I}}(t) + r_i^{\text{E}}(t). \tag{12}$$

The distinction between internal and external traffic is useful since it allows us to use two different methods for predicting their future arrival rates, and in turn a better prediction of the total arrival rate of the node, denoted by $\hat{r}_i(t)$:

$$\hat{r}_i(t) = \hat{r}_i^{\text{I}}(t) + \hat{r}_i^{\text{E}}(t), \tag{13}$$

where $\hat{r}_i^{\mathrm{I}}(t)$ and $\hat{r}_i^{\mathrm{E}}(t)$ are the predictions for the internal and the external traffic respectively. Next, the strategies for predicting these two will be described in more detail.

Predicting external traffic. When predicting the traffic arriving to the node directly from an external source the same strategy that was derived in the previous work will be used. The reason is that the timing-assumptions remain the same, as given by Table I. Naturally, one can use a more advanced way of predicting this traffic, however that is outside the scope of this paper. The main addition is the concept of using two distinct methods.

For the specific timing-assumptions of this paper, a sufficient prediction of the external traffic can be achieved trough linearization since the rate-of-change of the external traffic is assumed to be on a different time-scale than the time-delay Δ_i. Using linearization, the prediction of the external traffic is given by

$$\hat{r}_i^{\mathrm{E}}(t+\Delta_i) = r_i(t) + \Delta_i \cdot \frac{\mathrm{d}r_i(t+\Delta_i)}{\mathrm{d}t}. \tag{14}$$

Prediction of internal traffic. Due to the network structure of the nodes it is possible to achieve a very good prediction of the internal traffic, i.e., the traffic flowing between two different nodes in the network. In fact, this can be achieved by having every node that is sending traffic to the i-th node to also send its future predictions of this traffic to that node. If we denote the prediction of traffic from node i' to node i node as $\hat{r}_{i',i}^{\mathrm{I}}(t)$, the total prediction of the internal traffic arriving at node i can be written as

$$\hat{r}_i^{\mathrm{I}}(t) = \sum_{i' \in \mathcal{V}} \hat{r}_{i',i}^{\mathrm{I}}(t), \tag{15}$$

where it is the information about $\hat{r}_{i',i}^{\mathrm{I}}(t)$ that the i'-th node will send to node i. Node i can then simply sum these predictions to form a good prediction about the internal traffic that will arrive in the future.

Predicting $\hat{r}_{i',i}^{\mathrm{I}}(t)$ can then be done within the i'-th node, only using local information within that node. Furthermore, doing this can be decomposed into to smaller problems:

- predict the future service-rate $\hat{s}_{i'}(t)$, and
- predict the fraction of traffic routed to node i.

Predicting the future service rate of the i'-th node can be done in the same way as in the previous works, i.e., by assuming that it will be the minimum of the maximum service capacity and the incoming traffic of that node:

$$\hat{s}_{i'}(t+\Delta_i) = \min(s_{i'}^{\mathrm{cap}}(t+\Delta_i),\, \hat{r}_{i'}(t+\Delta_i)). \tag{16}$$

It should be noted that this prediction use information coming from the nodes sending traffic to the i'-th node through the use of $\hat{r}_{i'}(t+\Delta_i)$. It should also be noted that information about the maximum service capacity is known, since $s_{i'}^{\mathrm{cap}}(t+\Delta_i) = m_{i'}^{\mathrm{ref}}(t) \cdot (\bar{s}_{i'} + \xi_{i'}^{\mathrm{ub}})$. How $m_{i'}^{\mathrm{ref}}(t)$ is computed will be derived later, in Section III-B.

Finally, it remains to predict the fraction of outgoing traffic that will be routed to node i from node i'. By denoting the fraction of traffic that is currently routed from node i' to node i by $w_{i',i}(t)$, this can be predicted using the the same linearization as earlier:

$$\hat{w}_{i',i}(t+\Delta_i) = w_{i',i}(t) + \Delta_i \cdot \frac{\mathrm{d}w_{i',i}(t)}{\mathrm{d}t}. \tag{17}$$

Combining the two expressions of (16) and (17) yields the final expression for $\hat{r}_{i',i}^{\mathrm{I}}(t+\Delta_i)$ as

$$\hat{r}_{i',i}^{\mathrm{I}}(t+\Delta_i) = \hat{w}_{i',i}(t+\Delta_i) \cdot \hat{s}_{i'}(t+\Delta_i).$$

B. Service control

With the input prediction in place it is possible to achieve good service control. The service controller will use the predicted arrival rate to find the number of virtual machines that maximizes the utility metric (10). This is complicated somewhat by the fact that the performance of the virtual machines are uncertain, and might vary over time. To remedy this, we propose a feedback-loop from the actual performance of the VMs in order to compensate from any deviation from the respected performance. The work presented here builds upon the previous work in [2] and [1] with the addition of a theorem simplifying the service controller significantly when a node has a large number of VMs running.

The service controller developed in the previous works is given by:

$$m_i^{\text{ref}}(t) = \begin{cases} \lfloor \kappa_i(t) \rfloor, & \text{if } \lfloor \kappa_i(t) \rfloor \lceil \kappa_i(t) \rceil \geq \kappa_i^2(t) \\ \lceil \kappa_i(t) \rceil, & \text{else} \end{cases} \tag{18}$$

where $\kappa_i(t) = \hat{r}_i(t + \Delta_i)/\left(\bar{s}_i + \hat{\xi}_i(t)\right)$ is the real number of machines that is necessary to exactly match the predicted incoming traffic $\hat{r}_i(t)$. Recall that $\bar{s}_i$ is the expected service rate of an instance in the i-th node and that $\hat{\xi}_i(t)$ is the average machine uncertainty, i.e., the difference between the expected and the actual performance of the node.

The equation of (18) comes from trying to maximize the utility function (10). Since the exact number of VMs required to match the incoming traffic is given by $\kappa_i(t)$ it is necessary to either select $m_i^{\text{ref}}(t) = \lfloor \kappa_i(t) \rfloor$ and have slightly too little processing capacity (leading to packets being discarded) or to selecting $m_i^{\text{ref}}(t) = \lceil \kappa_i(t) \rceil$ and having slightly too much processing capacity (leading to wasting resources). The case deciding which to chose is such that it optimizes the utility function. For a complete derivation of (18), we refer to the earlier works.

Should one have a large number of virtual machines running in the nodes, the expression of (18) can be simplified significantly into $m_i^{\text{ref}}(t) = \lfloor \kappa_i(t) \rceil$ as shown in Theorem III.2 below.

Theorem III.1. *When the node is having a large number of virtual machines running $m_i^{\text{ref}}(t)$ can be computed as $m_i^{\text{ref}}(t) = \lfloor \kappa_i(t) \rceil$.*

Proof: Substituting $m_i^{\text{ref}}(t)$ for y and $\kappa_i(t)$ for x we can start from

$$y = \begin{cases} \lfloor x \rfloor, & \text{if } \lfloor x \rfloor \lceil x \rceil \geq x^2 \\ \lceil x \rceil & \text{else} \end{cases} \tag{19}$$

allowing us to write x as $x = \lfloor x \rfloor + r$, with $r \in [0, 1)$.

If $r = 0$ then $\lfloor x \rfloor = \lceil x \rceil = x$, implying that $y = x$. Otherwise, it follows that

$$\begin{gathered} \lfloor x \rfloor \lceil x \rceil \geq x^2 \\ \lfloor x \rfloor (\lfloor x \rfloor + 1) \geq (\lfloor x \rfloor + r)^2 \\ \lfloor x \rfloor^2 + \lfloor x \rfloor \geq \lfloor x \rfloor^2 + 2r \lfloor x \rfloor + r^2 \\ \lfloor x \rfloor \geq 2r \lfloor x \rfloor + r^2 \\ r \leq \frac{1}{2} - \frac{r^2}{2 \lfloor x \rfloor}. \end{gathered}$$

Hence, Eq. (19) can be rewritten as

$$y = \begin{cases} \lfloor x \rfloor, & \text{if } r \leq \frac{1}{2} - \frac{r^2}{2\lfloor x \rfloor} \\ \lceil x \rceil & \text{else} \end{cases}$$

which for large x, and then large $\lfloor x \rfloor$, becomes

$$y = \lfloor x \rceil .$$

■

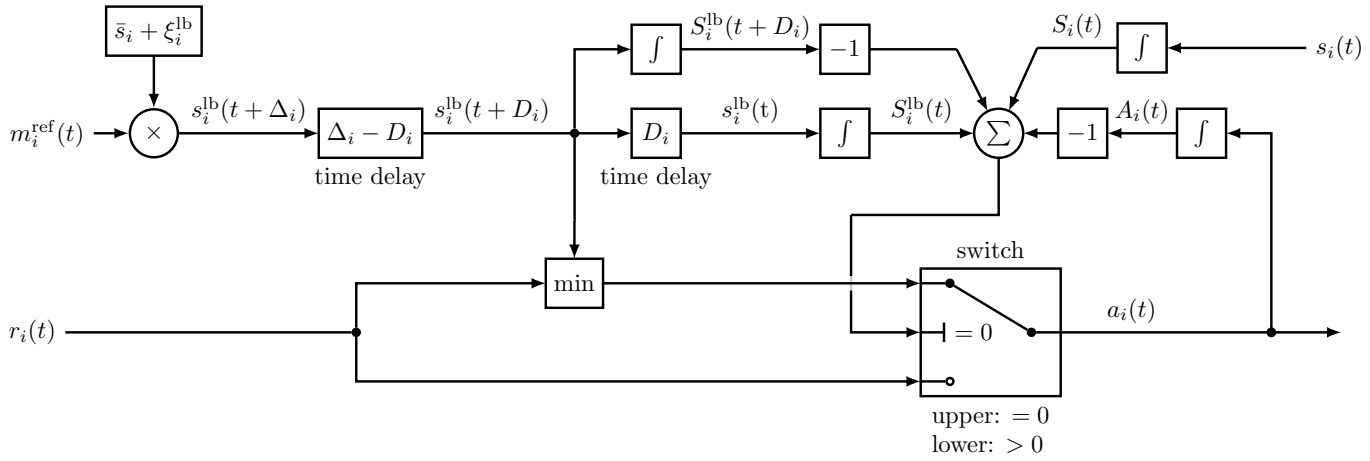


Fig. 4: The admission controller illustrated using a block diagram. The only computations necessary for the policy is a $\min(x, y)$-statement and an if-statement. The rest, e.g., the integration and the time-delays of the signals can be done continuously and require minimal computational effort.

C. Admission control

The task of the admission controller is to ensure that only packets that are guaranteed to be processed and pass through the node within the node deadline L_i are allowed to enter. This is done by controlling the rate by which packets are admitted into the node, the admission rate $a_i(t)$. The goal is naturally to admit as many packets as possible, while still guaranteeing that those admitted will meet the node deadline. In its essence, the admission controller ensures that there is never too large of a queue building up within the node. How large this queue can be is something that varies over time and depend on the arrival rates and the service rates of the node.

In Theorem III.2 we provide an optimal admission controller in the sense that it always admits as many packets as possible, while still guaranteeing that those admitted will meet the node deadline of D_i. Furthermore, as illustrated by the block diagram of the admission controller shown in Figure 4, the computation required for this admission policy can be computed instantly and in a continuous manner. This is very useful, as it allow the admission controller to continuously adapt the policy to changes of the node deadlines. In other words, the admission policy derived in Theorem III.2 allow one to dynamically change the node deadline over time $L_i(t)$. This is not something that will be used in this paper, but investigated in future work.

The admission policy derived in Theorem III.2 is given by:

$$a_i(t) = \begin{cases} r_i(t) & \text{if } A_i(t) < S_i(t) + S_i^{\text{lb}}(t+D_i) - S_i^{\text{lb}}(t) \\ \min(r_i(t),\ s_i^{\text{lb}}(t+D_i)) & \text{else.} \end{cases} \tag{20}$$

Some intuition behind this admission policy is that incoming packets are guaranteed to meet their deadlines as long as the upper bound on the delay is smaller than the node deadline, i.e. as long as $L_i^{\text{ub}}(t) > D_i$. It then follows from the definition of $L_i^{\text{ub}}(t)$, in (8), that the question whether $L_i^{\text{ub}}(t) > D_i$ can be expressed as:

$$S_i(t) + S_i^{\text{lb}}(t+D_i) - S_i^{\text{lb}}(t) > A_i(t).$$

Hence, as long as this inequality holds any incoming packet is guaranteed to meet its deadline, and should thus be admitted, (assuming that $r_i(t) < \infty$).

Should there instead be an equality in the expression above, implying that $L_i^{\text{ub}}(t) = D_i$, then care must be taken so that $L_i^{\text{ub}}(t)$ does not grow larger than D_i. Should this happen it becomes possible, due to the machine uncertainty, that a packet will miss its deadline. For this case, as shown in Theorem III.2, the largest possible admission rate is $a_i(t) \leq s_i^{\text{lb}}(t+D_i)$.

Theorem III.2. *The admission policy* (20) *will admit as many packets as possible while still ensuring that the admitted packets meet the node deadline.*

Proof: It follows from the definition of the upper bound of the delay $L_i^{\text{ub}}(t)$, given in (8), that incoming packets will meet their deadlines as long as

$$S_i(t) + S_i^{\text{lb}}(t + D_i) - S_i^{\text{lb}}(t) \geq A_i(t).$$

Should there be strict inequality, i.e.

$$S_i(t) + S_i^{\text{lb}}(t + D_i) - S_i^{\text{lb}}(t) > A_i(t),$$

any arriving packet is therefore guaranteed to meet its deadline. In such a case the function should admit packet at the same rate by which they arrive into the function, i.e. $r_i(t)$.

In the case of equality, incoming packet will be guaranteed to meet their deadlines as long as the following expression hold:

$$\frac{\partial}{\partial t}\{S_i(t) + S_i^{\text{lb}}(t + D_i) - S_i^{\text{lb}}(t) - A_i(t)\} \geq 0.$$

With an admittance rate $a_i(t)$ given by the admission policy of (20) the above expression can be written as

$$\frac{\partial}{\partial t}\{S_i(t) + S_i^{\text{lb}}(t + D_i) - S_i^{\text{lb}}(t) - A_i(t)\} = \underbrace{s_i(t)}_{\geq s_i^{\text{lb}}(t)} + s_i^{\text{lb}}(t + D_i) - s_i^{\text{lb}}(t) - \underbrace{a_i(t)}_{\geq s_i^{\text{lb}}(t+D_i)} \geq s_i^{\text{lb}}(t) + s_i^{\text{lb}}(t + D_i) - s_i^{\text{lb}}(t) - s_i^{\text{lb}}(t + D_i) = 0.$$

Here one should note that adopting a different admission policy where $a_i(t) < s_i^{\text{lb}}(t + D_i)$ implies that that the above inequality would be strictly greater than 0, thus discarding more packets than necessary.

Assuming the system starts with empty queues at time $t = 0$ it follows from the definition of $S_i(t)$ and $A_i(t)$ that $S_i(t) + S_i^{\text{lb}}(t + D_i) - S_i^{\text{lb}}(t) \geq A_i(t)$, since $S_i(0) = A_i(0) = 0$ and $S_i^{\text{lb}}(t)$ is non-decreasing.

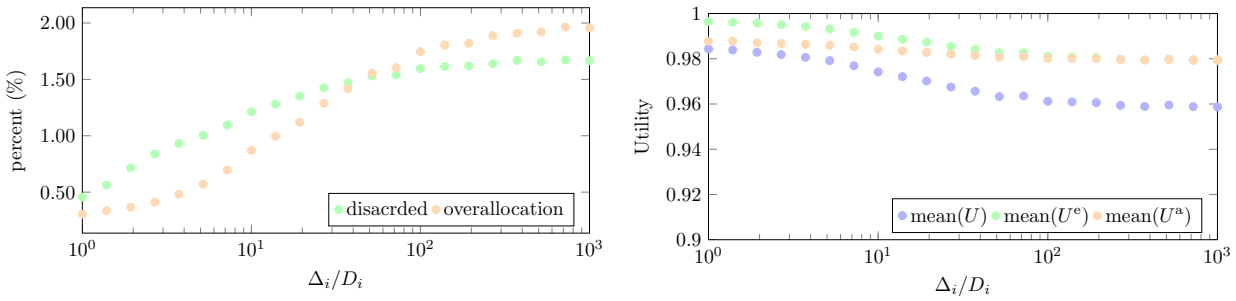
The system will therefore never end up in a state where $L_i^{\text{ub}}(t) > D_i$, which ensures that the incoming packets, and thus also the outgoing packets, will always meet their deadlines. ■

D. Selection of node deadlines

When addressing the problem of selecting the node deadlines for the nodes in the network one must know how different choices affect the utility of the system. Naturally, having an unnecessarily low node-deadline would cause unnecessarily many packet to be discarded for that node. Furthermore, while greatly improving the complexity for the node, adding a node deadline means introducing some *slack* into the system. By slack we mean that the sum of the local node deadlines for the nodes along a path might be lower than the end-to-end deadlines of that path. It is not possible to get around the fact that one introduces slack using this simplification of having a single node deadline, so the questions is rather, *where should the slack be introduced?*

To gain some understanding in how different node deadlines affect the amount of packets that are discarded a thorough analysis was made. Using the controllers presented earlier in this section a large number of Monte Carlo simulations was performed to investigate this. These simulations suggest that there is a relationship between the number of discarded packets and the ratio Δ_i/D_i, as illustrated in Figure 5. For the simulations resulting in this figure it should be noted that every data-point correspond to the mean value of that metric achieved over 1,000 simulations (using the method described in Section IV-A).

Intuition behind Δ_i/D_i. The intuition behind the ratio of Δ_i/D_i is that it captures the innate difficulty of controlling the node. The ratio gives insight in the difference of how long it takes to change the number of VMs that are running, and the node deadline. A smaller ratio means that it is easier to



(a) **(b)**

Fig. 5: Figure illustrating how the ratio Δ_i/D_i affects the performance of a node as well as the amount of discarded packets and overallocation of resources. The intuition behind these results is that the larger the ratio, the harder it is to have good control of the node since one has to predict the arrival rates further into the future.

react to changes of the arrival rate. In fact, a ratio smaller than 1 implies that there is always time to react to such changes. As the ratio grows larger, it is no longer possible to react to the changes in the arrival rate. Instead, it becomes necessary to predict the future input rates. Finally, the larger the ratio, the longer into the future one must predict the arrival rates, hence the harder it becomes to control the node.

Setting up the optimization problem. Using the insights of the relationship between the ratio of Δ_i/D_i and the performance of the system, it is natural to set up the node deadline assignment as the following problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in \mathcal{V}} \Delta_i/D_i \\ \text{subject to} \quad & \sum_{i \in p_j} \delta_{j,i} \cdot D_i \leq \mathcal{D}_j \quad j = 1, \ldots, f \\ & D_i \geq 0 \quad \forall i \in \mathcal{V} \end{aligned} \tag{21}$$

where $\delta_{j,i}$ indicates how many timed path j goes through node i and Δ_i indicates the time-overhead necessary to change the number of VMs in the i-th node. One should note that the optimization problem (21) is convex and can thus be solved using disciplined cone programming or by standard methods such as Lagrange multipliers.

Solve with Lagrange multipliers. As mentioned earlier one can solve the optimization problem (21) using Lagrange multipliers. Disregard the final constraint of (21), which is possible due to the convex nature of the problem, one arrives at the following Lagrangian:

$$L = \sum_i \frac{\Delta_i}{D_i} + \sum_j \mu_j \cdot (\sum_i \delta_{j,i} D_i - \mathcal{D}_j) \tag{22}$$

where D_i and μ_j are the primal and dual optimization variables. The Karush-Kuhn-Tucker conditions for (22) are:

$$\begin{aligned} & \frac{\partial L}{\partial D_i} = -\frac{\Delta_i}{D_i^2} + \sum_j \mu_j \delta_{j,i} = 0 \quad \forall i = 1, \ldots, n \\ & \mu_j \cdot (\sum_i \delta_{j,i} D_i - \mathcal{D}_j) = 0 \qquad \forall j = 1, \ldots, f \\ & \mu_j \geq 0 \qquad \forall j = 1, \ldots, f \end{aligned} \tag{23}$$

It should be noted that when solving (23) one will likely end up with many possible cases for μ_j and D_i, however, only a single set of parameters will yield an optimal value for (22), which can be denoted as (D_i^*, μ_j^*).

IV. Evaluation

The evaluate the performance of the performance of the generalized AutoSAC derived in this paper we run a large Monte Carlo simulation of a network of cloud functions, similar to the one illustrated in Figure 2. For every simulation the properties of the network was randomly generated, as described in detail in Section IV-A, and as input data to simulate the traffic moving through the network we used a real traffic trace from the Swedish university network (SUNET) was used. With this set-up, the generalized AutoSAC was compared to what is currently being used in industry today. These industry-methods are: **a)** *dynamic auto-scaling* (DAS), and *b) dynamic overallocation* (DOA), both described in Section IV-B. We also chose to augment both of these industry method with the admission controller developed in this paper, since neither DAS nor DOA has one.

Results. The result of the Monte Carlo simulation is shown in Figure 6. The left part show the comparison of the average utility, efficiency, and availability. The right part illustrate the same result, but by instead showing the fraction of packets that are discarded as well as how much overallocation each method cause.

One can see that AutoSAC is close to optimal in the average utility, availability, and efficiency. As expected, DAS has an efficiency in the range of 0.8–0.9, since those are the thresholds by which it bases its auto-scaling on. DOA achieves an efficiency of around 0.85 which is also expected due to the amount of overallocation it has. The reason for their poor average utility however, is due to the lack of a admission controller. Without one a queue will build up every time there is a lack of the amount of available resources. This increases the latency, causing packets to miss their deadlines which in turn results in a low availability. One can see that by augmenting DAS and DOA with the admission controller developed in Section III-C the availability is significantly increased. The admission controller is only able to achieve this by discarding some of the packets, but by looking at the right part of Figure 6 one realize that it is only a very small fraction of the packets that are actually discarded, so it is a sacrifice well worth making.

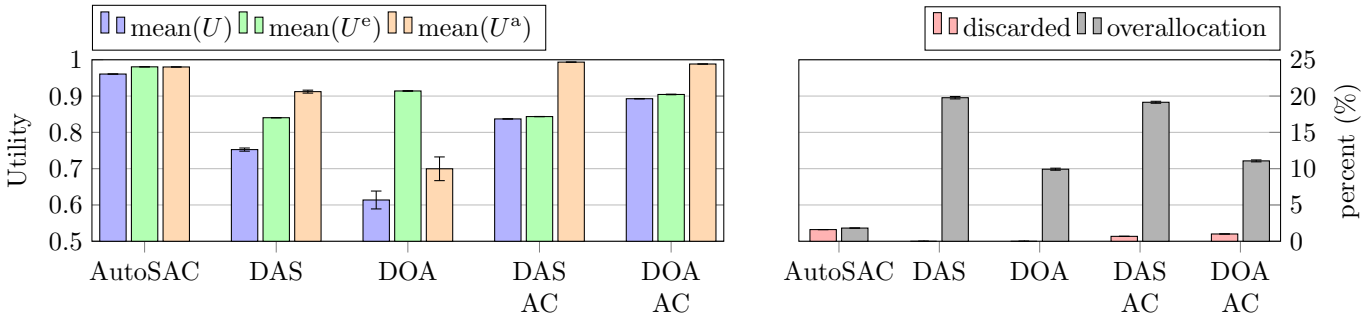


Fig. 6: Through a large number of simulations of the network depicted in Figure 2 (where for each of the simulations the properties of each node was randomly generated, and where the incoming traffic for each of the paths was taken from a real traffic trace from the Swedish university network) the performance of the automatic service- and admission controller (AutoSAC) developed in this paper was compared with what is currently being used in industry. The methods AutoSAC was compared against was dynamic auto-scaling (DAS) and dynamic overallocation (DOA). Neither DAS nor DOA has admission controller so they were augmented with the one developed in this paper. In the left figure one can see the result of the average utility, efficiency, and availability for each of the five methods, and in the right figure the fraction of the incoming packets that are discarded and the average amount of overallocation of resources.

A. Simulation method

For each of the simulations used as evaluation the parameters of the VNFs were randomly generated. Furthermore, the path-wise end-to-end deadlines were randomly chosen from an interval of $[0.25,\ 0.5]$ seconds. These deadlines were then decomposed into local node deadlines using the convex problem given in (21). The input rate for each path was randomly chosen from a data-base of traffic data gathered from SUNET. The traffic was scaled to have a peak of $10\,000\,000$ pps, see Figure 7a for data from one of the simulations.

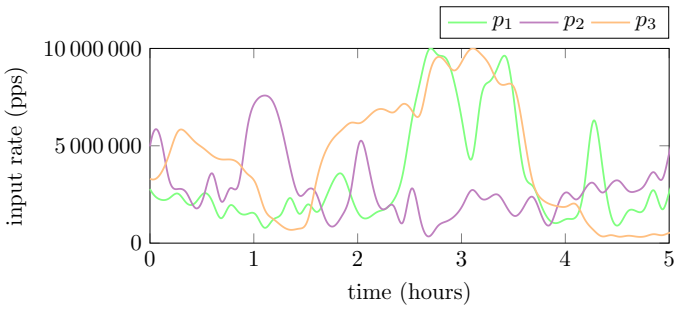
The nominal service rate $\bar{s}_i$ was randomly chosen from the interval $[100\,000, 200\,000]$ pps, implying that a node would need about 5-10 instances to match the traffic coming from a single path (and naturally more the more paths that flow through the node). The time-overhead Δ_i was uniformly chosen from the interval $[15,\ 60]$ seconds, leading to every node having a Δ_i/D_i ratio of about 30–200. Finally, the bounds for the machine uncertainty was chosen randomly from the intervals $\xi_i^{\text{lb}} \in [-0.3 \cdot \bar{s}_i,\ 0]$ and $\xi_i^{\text{ub}} \in [0,\ 0.3 \cdot \bar{s}_i]$. During the simulation the machine uncertainty would then be chosen randomly from within these bounds: $\hat{\xi}_i(t) \in [\xi_i^{\text{lb}},\ \xi_i^{\text{ub}}]$ every time a new instance was started/stopped in order to simulate the load disturbance generated from not knowing the performance of the physical machine that the instance is launched on. This led to the step-wise behavior shown in Figure 7c. When the number of instances did not change for a node, the machine uncertainty was simulated as a stochastic process leading to the small drift shown in Figure 7c.

B. Comparison with state-of-the art

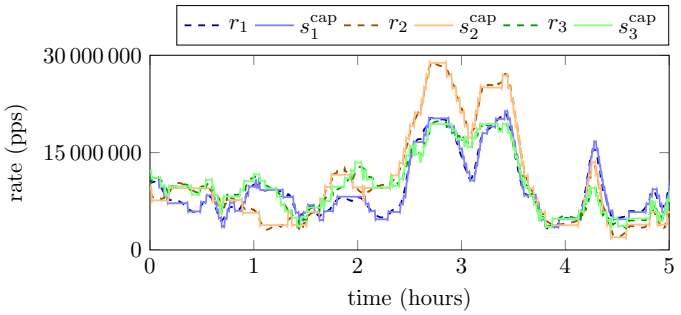
As mentioned earlier, in order to evaluate the performance of the generalized version of AutoSAC it is compared against two methods considered standard in the industry today. These two methods are *dynamic auto scaling* (DAS) and *dynamic overallocation* (DOA). Both of these methods are briefly described below. Since neither of these two methods use an admission controller, we augment them with the the one developed in this paper, resulting in two additional methods: DAS+AC and DOA+AC. Each method was simulated using a Monte Carlo simulation with 1,000 runs performed according to Section IV-A.

Dynamic auto-scaling (DAS). This is the auto-scaling method offered to customers of Amazon Web Services, [18]. It is a purely reactive method and is based by having the users monitoring a specific metric (e.g., CPU utilization) of their VMs using CloudWatch. The user then specifies two thresholds on which the auto-scaling is based upon. For this evaluation the efficiency metric $u_i^{\text{e}}(t)$ was used as the auto-scaling metric with the following thresholds:

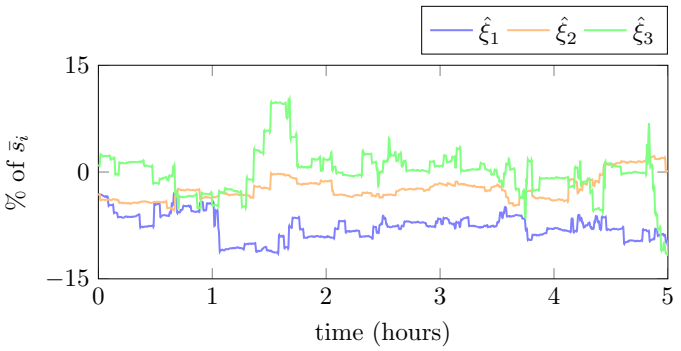
$$\begin{cases} \text{add a VM} & \text{if } u_i^{\text{e}}(t) > 0.9, \\ \text{remove a VM} & \text{if } u_i^{\text{e}}(t) < 0.8. \end{cases}$$



(a) Input traffic for each of the three paths in the simulated network. One can see that there are large deviations in the amount of incoming traffic, thus forcing the VNF nodes in the network to quickly scale up/down.



(b) The incoming traffic for each of the VNF nodes in the network (the dashed lines) highlight the need for elasticity of the amount of allocated resources to the nodes. In solid lines one can see the maximum service capacity for each node when the nodes use AutoSAC.



(c) The average machine uncertainty, i.e. the difference between the expected and true service performance of the virtual machines, illustrated for each node in the network. It is normalized with respect to the nominal service capacity of one instance, implying that if $\hat{\xi}_i(t) = -10\%$ then the i-th node would need 10% more instances in order to match the incoming traffic load.

Fig. 7: Simulation data from one of the many simulations used for evaluation and comparison in Section IV-B.

Dynamic overallocation (DOA). A downside with DAS is that it only uses feedback to control the number of instances it needs. With a large ratio between the time-overhead and the local node deadline it becomes very difficult to control solely based on feedback. An alternative approach commonly used in industry is to instead use dynamic overallocation where one measures the input to each function and allocates virtual resources such that there is an expected overallocation of 10%.

V. Summary

In this paper we derive a general mathematical model for NFV graphs on top of which multiple forwarding paths are defined. The model is used to synthesize an automatic service- and admission controller (AutoSAC) with the goal of controlling the amount of resources allocated to the virtual network functions in the NFV graph. The goal is to control them in a way such that an end-to-end performance for the forwarding paths can be ensured. AutoSAC is evaluated and compared against four other methods commonly used in the cloud industry today. The evaluation, based on a large Monte Carlo simulation, shows that AutoSAC is able to outperform the other method and achieve a good performance, both in availability and efficiency. The evaluation also illustrates the importance of having good admission control.

Interesting possibilities for future work would be to extend the model to include dynamics in the forwarding paths, i.e. to allow for new paths and to allow existing paths to leave the network. This is made possible with the new admission controller presented in this work. It makes it possible to continuously change the node deadlines, while still guaranteeing that every admitted packet will still meed the node deadline (at the time of admission). The future work should thus be focusing on how to change the node deadlines in order to allow the network to accept new packet-flows with new end-to-end deadlines.

Source code Traffic data and code for simulations: https://github.com/vmillnert/GLOBECOM18simulation.

References

[1] V. Millnert, E. Bini, and J. Eker, "AutoSAC: automatic scaling and admission control of forwarding graphs," *Annals of Telecommunications*, vol. 16, no. 3, pp. 15–12, Aug. 2017.

[2] V. Millnert, J. Eker, and E. Bini, "Dynamic control of NFV forwarding graphs with end-to-end deadline constraints," in *ICC 2017 - 2017 IEEE International Conference on Communications*. IEEE, 2017, pp. 1–7.

[3] T. Lin, Z. Zhou, M. Tornatore, and B. Mukherjee, "Optimal Network Function Virtualization Realizing End-to-End Requests," in *GLOBECOM 2015 - 2015 IEEE Global Communications Conference*. IEEE, 2014, pp. 1–6.

[4] W. Shen, M. Yoshida, T. Kawabata, K. Minato, and W. Imajuku, "vConductor: An NFV management solution for realizing end-to-end virtual network services," in *2014 16th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2014, pp. 1–6.

[5] Y. Li, L. T. X. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE INFOCOM 2016 - IEEE Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[6] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *2010 11th IEEE/ACM International Conference on Grid Computing (GRID)*. IEEE, 2010, pp. 41–48.

[7] A. Leivadeas, M. Falkner, I. Lambadaris, and G. Kesidis, "Resource Management and Orchestration for a Dynamic Service Chain Steering Model," in *GLOBECOM 2016 - 2016 IEEE Global Communications Conference*. IEEE, 2016, pp. 1–6.

[8] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Dynamic network service optimization in distributed cloud networks," in *IEEE INFOCOM 2016 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 300–305.

[9] G. Faraci, A. Lombardo, and G. Schembra, "A building block to model an SDN/NFV network," in *ICC 2017 - 2017 IEEE International Conference on Communications*. IEEE, 2017, pp. 1–7.

[10] Y. Ren, T. Phung-Duc, J.-C. Chen, and Z.-W. Yu, "Dynamic Auto Scaling Algorithm (DASA) for 5G Mobile Networks," in *GLOBECOM 2016 - 2016 IEEE Global Communications Conference*. IEEE, 2016, pp. 1–6.

[11] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 50, pp. 117–134, Apr. 1994.

[12] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," *Real-Time Systems*, vol. 30, no. 1–2, pp. 105–128, 2005.

[13] J. L. Lorente, G. Lipari, and E. Bini, "A hierarchical scheduling model for component-based real-time systems," in *Proceedings of the 20-th International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, Apr. 2006.

[14] M. Ashjaei, S. Mubeen, M. Behnam, L. Almeida, and T. 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)*, Aug. 2016, pp. 1–11.

[15] D. Natale and Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings Real-Time Systems Symposium*. IEEE Comput. Soc. Press, 1994, pp. 216–227.

[16] S. Jiang, "A Decoupled Scheduling Approach For Distributed Real-Time Embedded Automotive Systems," in *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. IEEE, pp. 191–198.

[17] P. Leitner and J. Cito, "Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Transactions on Internet Technology*, vol. 16, no. 3, pp. 1–23, Aug. 2016.
[18] (2016, 10). [Online]. Available: https://aws.amazon.com/documentation/autoscaling/