Elastic Function Chain Control for Edge Networks under Reconfiguration Delay and QoS Requirements

Michele Berno* Flavio Esposito† Michele Rossi*
*Dept. of Information Engineering, University of Padova, Italy
†Dept. of Computer Science, Saint Louis University, Saint Louis (MO), USA
email: michele.berno@dei.unipd.it, flavio.esposito@slu.edu, rossi@dei.unipd.it

***Abstract*—Network Function Virtualization (NFV) allows network providers to reconfigure their edge processing infrastructure in an *online* fashion, to adapt it to the changing traffic demands (intensity and type of computation requests). In this work, we consider the Virtual Network Function Placement and Chaining (VNFPC) problem, whose aim is to elastically deploy services (i.e., chains of multiple Virtual Functions, VFs) through three phases: function *placement*, *assignment*, and *chaining*. For this problem, a *predictive* control framework is proposed to solve these three phases jointly by horizontally scaling VF instances, adapting their number to current and predicted demands, while ensuring that flows' Quality of Service (QoS) requirements (latency) are met. Our technique accounts for the delays and costs incurred in reconfiguration operations and uses a Gaussian Mixture Model, trained with real traces collected by base stations across the city of Milan (Italy), to estimate future computing demands. The proposed predictive control method is tested against a heuristic policy for several meaningful metrics, achieving up to 99% less edge control overhead and allowing a reduction of the blocking probability by 95% with respect to the heuristic (for the same energy consumption).**

I. Introduction

The emergence of network function virtualization (NFV) and software-defined networking (SDN) enables providers to deploy network services in the form of interconnected software functions instantiated over commercial off-the-shelf servers. These services can be deployed on computationally large cloud environments and resource-limited edge devices. The *elastic* allocation of computing resources makes it possible to reduce capital and operational expenses, while meeting Quality of Service (QoS) requirements. The set of services includes network management (control plane) and user applications (user/data plane), composed and chained together via Virtualized Functions (VFs).

To deploy such services, providers need to interconnect and place VF instances, solving the so-called Virtual Network Function Placement and Chaining (VNFPC) problem. The goal of this problem is to ensure that ordered flows of computing tasks are assigned to a suitable (possibly optimal) processing chain described through a Directed Acyclic Graph (DAG).

The VNFPC problem can be decomposed into three phases: *(i)* function ***placement***, *(ii)* ***assignment***, and *(iii)* ***chaining***. With the *placement phase*, the system determines the number of Virtual Function Instances (VFIs) that are necessary to meet the current computation demand and where VFIs should be placed. The *assignment phase* selects which placed VFI will be in charge of which flow (note that VF instances can be shared by flows). Lastly, VFIs are *chained*: this process consists of creating paths that interconnect the VFs that were placed and assigned in the previous phases while guaranteeing that QoS requirements are met (in this work, that the end-to-end latency experienced by the flows does not exceed the flow-specific deadlines). Efficient VF instance placement and processing chain assignment are key to ensure that resource allocation is carefully orchestrated, preventing over- or under-provisioning of resources.

Our contribution. Our current focus is to control the edge part of the network infrastructure, providing means for the online assignment, placement, and chaining of VFs under QoS constraints and in the presence of resource-constrained devices. A distinctive trait of our work is that our algorithm has *predictive capabilities*, i.e., future computing requests are forecast and used to solve the VNFPC at runtime, while meeting delay constraints. Also, existing works on this subject usually ignore delays, i.e., virtual machines (VM) startup time, and costs, i.e., the energy consumption of reconfiguration operations. For example, starting up a VM can require up to 2 minutes [1]. The main contributions of the present paper are: *i)* we expand the VNFPC problem to dynamic settings (multi-period allocation), accounting for reconfiguration costs and delays, with the objective of finding the right adaptation/reconfiguration trade-off; *ii)* we train Gaussian Mixture Models (GMM) to predict real-world flow intensities; *iii)* we propose a novel Model Predictive Control (MPC) approach to solve the multi-period VNFPC problem, tackling all the three phases *jointly*, in an *online* fashion and exploiting GMM-based demand forecasts; *iv)* we devise a tunable heuristic approach with low complexity, which does not exploit demand forecasts and solves the placement phase separately from the other two; *v)* we evaluate the performance of *iii)* and *iv)* on base station activity traces extracted from a publicly available dataset.

We remark that the heuristic control policy solves the assignment and chaining phases optimally for the given available (deployed) virtual resources, but the placement phase is solved independently from the other two and without using information about future demands. Such policy is designed to quantify the benefits of joint allocation and prediction of computing demand. Our results reveal that the proposed predictive approach can achieve up to 99% less edge control overhead and allows reducing the blocking probability by up to 95% with respect to the heuristic policy, while maintaining

This work has been partially supported by NSF awards CNS-1647084, CNS-1836906, CNS-1908574, by the Italian PRIN project no. 2017NS9FEY and by MIUR (Italian Ministry of Education, University and Research) through the initiative "Departments of Excellence" (Law 232/2016).

978-1-7281-1035-6/20/$31.00 ©2020 IEEE
DOI 10.1109/MobileCloud48802.2020.00018

the energy consumption of the edge network at a similar level.

The paper is organized as follows. In Section II, we summarize the related work. Our system is described in Section III, in Section IV we formulate our proposed solution strategies, and in Section V we show some selected numerical results. Future research avenues are discussed in Section VI.

II. Related Work

The VNFPC problem bears some similarities with the Virtual Network Embedding (VNE) problem, which investigates how to deploy virtual network requests atop a physical substrate [2]–[4]. Usually, VNFPC problems consider a *two-level mapping* of resources, namely, mapping processing chain requests (flows) onto VF instances, that are in turn mapped onto a physical network, while VNE assumes *one-level resource mapping*, i.e., virtual network requests on a physical network [2], [5]. The authors of [6] formulate the problem of VNF placement (without chaining nor QoS requirements) considering a two-level resource mapping as a generalization of Facility Location and Generalized Assignment Problems (FLP and GAP). The authors of [5], [7] investigate how to ensure that resources are correctly allocated and orchestrated, preventing their over- or under-provisioning and keeping end-to-end delays comparable to those observed in traditional middlebox-based networks. In [8], hybrid networks also containing physical hardware (middleboxes) are considered. In [7], a proof of the NP-hardness of the VNFPC problem is given. The authors of [9] solve the VNFPC problem with one-level resource mapping for distributed cloud networks, formulating it as a novel multi-commodity-chain (MCC) queuing system. The authors of [10] also consider the delay and cost associated with reconfiguration operations.

To best utilize resources, NFV providers need to dynamically scale the VNF deployments and reroute traffic demands for their customers. There are several techniques to implement resource elasticity (for example, see [11] and [12]). The two most prominent mechanisms to achieve it are: *i)* ***horizontal scaling*** of VFIs and *ii)* ***vertical scaling***. The first consists in the creation/removal (scale-out/in) of multiple virtual function instance replicas, while the latter scales up or down (increasing or reducing) the amount of computing resources that are allocated to a particular VFI based on varying demands.

While most existing works are reactive, the authors of [13] seek a proactive approach to provision new instances for overloaded VFIs ahead of time-based on the estimated flows intensity. [14] optimizes the amount of computation resources that are purchased across different timescales from the public cloud. [15] introduces an energy-aware consolidation strategy based on predictive control for cloud infrastructures.

III. System Model

A. Network Infrastructure (Substrate)

The underlying physical infrastructure is described by the attibuted and directed graph $G = (\mathcal{N}, \mathcal{E}, \mathcal{A}^N, \mathcal{A}^E)$, where the set $\mathcal{N}$ collects physical network nodes, while the set $\mathcal{E}$ contains the existing (directed) physical links connecting them. The sets $\mathcal{A}^N = \{A_i^N = (T_i, V_i),\, i \in \mathcal{N}\}$ and $\mathcal{A}^E = \{A_{ij}^E = (b_{ij}, d_{ij}),\, (i,j) \in \mathcal{E}\}$ contain nodes' and links' attributes, respectively. Node type attributes $T_i \in \{\text{access}, \text{router}, \text{edge}, \text{cloud}, \text{sink}\}$ induce a partition over set $\mathcal{N}$, i.e., $\mathcal{N}^{\text{a}} \cup \mathcal{N}^{\text{r}} \cup \mathcal{N}^{\text{e}} \cup \mathcal{N}^{\text{c}}$. Every access node is connected to a sink node collecting rejected processing requests through a 0-latency uncapacitated link. The sink is an artificial node that is added to the model to track computation requests that are to be rejected due to the lack of resources. V_i defines the maximum number of parallel Virtual Function Instances (VFIs) supported by node $i \in \mathcal{N}$ (e.g. number of CPU cores). Note that, for router nodes it holds $V_i = 0$, while for cloud and sink nodes we assume $V_i = \infty$, i.e., they always have enough available resources for the assigned processing chains. Lastly, link attributes b_{ij} and d_{ij} respectively define capacity and delay of link $(i,j) \in \mathcal{E}$.

TABLE I: List of symbols used in the paper

Name	Definition
G	Network infrastructure (directed) graph
$\mathcal{N}$, $\mathcal{A}^N$	Physical nodes and their attributes
$\mathcal{E}$, $\mathcal{A}^E$	Physical links and their attributes
T_i , V_i	Node type and max. parallel VFI
b_{ij}, d_{ij}	Bandwidth and delay of phy. link $(i,j) \in \mathcal{E}$
k, Δk	Time slot index and duration
$\mathcal{M}$, $\mathcal{S}$	Sets of Virtual Functions and Processing chains
$q = (a_q, s_q, r_q)$	PC of type s_q originated in $a_q \in \mathcal{N}^a$ (flow r_q)
$\mathcal{Q}_k$	Processing chain requests generated at time k
$G^{Pc}(s_q)$	DAG associated with PC $q = (a_q, s_q, r_q)$
$\mathcal{N}_{s_q}^{Pc}$, $\mathcal{E}_{s_q}^{Pc}$	Stages and virtual paths of PC q
$x_i^{q,l}(k)$	(0/1) PCq's l-th stage assigned to ES i
$x_{ij,uv}^{q}(k)$	(0/1) virtual path $(u,v) \in \mathcal{E}_{s_q}^{Pc}$ to link $(i,j) \in \mathcal{E}$
$y_{i,m}(k)$	(Integer) Type m VFI being deployed on $i \in \mathcal{N}^e$
$\tilde{y}_{i,m}(k)$	(Integer) Type m VFI being removed from $i \in \mathcal{N}^e$
D_s	Tolerated E2E-latency for a type-s ($\in \mathcal{S}$) PC

B. Virtual Function Instances and Processing Chains

Computational resources distributed over the network (at the edge and cloud) can be used to execute computing tasks. A Directed Acyclic Graph (DAG) is used to describe the set of ordered Virtualized Functions (VFs) that need to be applied to an offloaded task requesting a given service. We refer to flows of tasks requesting specific services as Processing Chains (PC). As shown in Fig. 1, we assume a two-level mapping between processing chain requests (i.e., demands) and physical resources: *i)* Virtual Function Instances (VFIs) are dynamically deployed on physical nodes and *ii)* Processing chain requests are dynamically assigned to active VFIs and links (i.e., routing-awareness). Our main focus is on controlling the edge infrastructure. Thus, we assume that the cloud has always enough VFIs to support the incoming load. As anticipated, for computing-enabled edge nodes $i \in \mathcal{N}$, V_i upper bounds the number of supported parallel VFIs. Usually, such V_i is small for edge networks, and this renders an optimal orchestration of VFs and their chaining especially important. The system evolves at discrete increments $k = 0, 1, \ldots$ of length Δk.

Virtual functions (VF): VFs can offer type m processing, where $m \in \mathcal{M}$ (finite). The set $\mathcal{A}^{\text{vf}} = \{A_m^{\text{vf}} = (R_m^{\text{vf}}, \tau_{i,m}^{\text{vf}}),\, m \in \mathcal{M}\}$ specifies VF attributes. R_m^{vf} is the processing capacity of a single instance of m-type

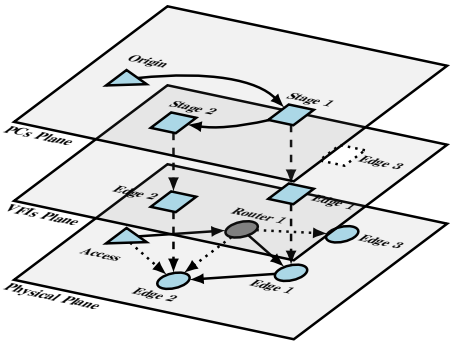


Fig. 1: **Edge physical infrastructure and virtual abstraction layers.** Processing stages of a PC are associated with available resources at the underlying physical infrastructure. Edge server 1 and 2 need to have the requested virtual function instances deployed on them to support the PC request. Also, VFIs need to have sufficient computational resources available.

VF, measured in number of tasks (or Mb) per time slot. Instead, $\tau^{\mathrm{vf}}_{i,m}$ indicates the average execution time of tasks requiring a type-m processing at node i.

Processing chains (PC): a processing chain demand $q = (a_q, s_q, r_q)$ is a flow of tasks with an intensity of r_q tasks per time slot, requesting a given processing service $s_q \in \mathcal{S}$ and injected by access node $a_q \in \mathcal{N}^a$. The set $\mathcal{S}$ collects the processing services available (assumed to be finite). We denote the PC requests present in the system during time step k by the set $\mathcal{Q}_k$. A service type s is characterized by its *processing graph* (DAG) $G^{Pc}(s) = (\mathcal{N}^{Pc}_s, \mathcal{E}^{Pc}_s)$ that specifies *i)* the $L_s \geq 1$ *processing stages* composing the associated service s ($l \in \mathcal{N}^{Pc}_s$), *ii)* all the directed virtual paths connecting these stages, contained in $(u, v) \in \mathcal{E}^{Pc}_s$. Each processing stage $l \in \mathcal{N}^{Pc}_s$ indicates the type of processing (VF) required among the existing ones ($\mathcal{M}$). Moreover, each service imposes a strict deadline of D_s [millisecond] on the maximum tolerated latency that a packet requesting service s can experience, from the access node to the last stage in $l \in \mathcal{N}^{Pc}_s$. In this work, we focus on linear sequences (chains) of processing stages, leaving more complex graphs as a future work. PC requests have to be placed during the time slot of their arrival, i.e., they cannot be backlogged. Therefore, in the case of insufficient resources they are rejected.

C. Decision Variables (Controls)

Managing VFIs requires time, and the limited computing capabilities of edge devices require to wisely decide when, where, and which type of VF to deploy, remove, or keep active. Moreover, the resource scheduler must consider the latency introduced by the deployment of a new VFI in task assignment and scaling operations, towards supporting time-sensitive applications. Hence, at each time slot $k = 0, 1, \ldots$, the VNFPC problem involves the following interconnected phases:

i) **VFI scaling agent:** adapts the number of deployed VFI instances within the edge to demand fluctuations (both across access nodes and time slots). Variables $y_{i,m}(k)$ and $\tilde{y}_{i,m}(k)$, both in $\{0, 1, \ldots, V_i\}$, respectively represent the number of VF instances of type m to be deployed on and removed from node i. Note that the deployment or removal of any VFI at time k will be into effect at time $k + 1$, i.e., a one-slot lag.

ii) **PC scheduler:** assigns PCs' stages ($\mathcal{N}^{Pc}_s$) to already deployed VFIs. The binary variables $x^{q,l}_i \in \{0, 1\}$, $l \in \{1, \ldots, L_q\}$, $q \in \mathcal{Q}_k$, $i \in \mathcal{N}$, indicate whether stage l of PC q is assigned to node i.

iii) **Chaining agent:** has to build a suitable path connecting access nodes with the ordered sequence of physical nodes assigned to each stage of the considered PCs. The chaining decision $x^q_{ij,uv}$ assigns physical link $(i, j) \in \mathcal{E}$ to the virtual path $(u, v) \in \mathcal{E}^{Pc}_{s_q}$ (of the DAG $G^{Pc}(s_q)$), for every PC $q \in \mathcal{Q}_k$.

D. Processing Chain Generation Process

The generation process of processing chain requests of type $s \in \mathcal{S}$ is dynamic in time and could also vary across access nodes. These exogenous processes are described through two independent components, namely,

i) **arrival process** $\delta_{a,s}(k)$, $k = 0, 1, \ldots$ defines the time slots in which a processing chain request of type s is generated by access node $a \in \mathcal{N}^a$. Therefore, $\delta_{a,s}(k) = 1$ indicates that a processing chain request has arrived at the system through access node a in time slot k. In this work, to characterize the (time-correlated) arrival process we use Markov Chains (MCs). By tuning MC transition probabilities it is possible to obtain different generation behaviors. For example, a two state MC (zero/one arrivals) with transition probabilities p_{01}, p_{00}, p_{11}, and p_{10} exhibits an average arrival rate per time slot of $\lambda = p_{01}/(p_{01}+p_{10})$, and an average burst length of $b = 1/p_{10}$ (subsequent slots where arrivals occur).

ii) **Flow intensity** $r_{a,s}(k)$ specifies the flow intensity (comp. tasks per time slot) for each type-s service and access node $a \in \mathcal{N}^a$ ($r_q = r_{a_q,s_q}(k)$). We use real base station activity traces (SMS, call, and Internet traffic) to model flow intensity processes (varying across access nodes and over time). In this way, we assume that processing chain flow intensity is proportional to the traffic experienced by an access node (base station). More details about this are given in Section V-A.

E. Exogenous Process Forecast

To exploit correlated patterns in the exogenous processes, we use predictors for components *i)* and *ii)* of Section III-D.

i) **Arrival process predictor** knows the statistical model governing PC arrival times for each access node $a \in \mathcal{N}^a$ and PC type $s \in \mathcal{S}$. PC arrival time predictions $\hat{\delta}_{a,s}(p)$ for the future time slots $p = k + 1, \ldots, k + W_{\mathrm{ph}} - 1$ are obtained unrolling the corresponding Markov Chain over these time slots, starting from the current state $\delta_{a,s}(k)$. W_{ph} is the considered prediction horizon (see Section IV-D for details). In the following, we refer to this predictor with the letter C.

ii) **Flow intensity predictor** is here implemented using Gaussian Mixture Models (GMMs), which are trained to predict flow intensities $\hat{r}_{a,s}(p)$ for future time slots $p = k + 1, \ldots, k + W_{\mathrm{ph}} - 1$. After being trained in a supervised manner, GMMs take as input the sequence of past flow intensity measurements $r_{a,s}(k - W_{\mathrm{lb}}), \ldots, r_{a,s}(k)$ of length W_{lb} (*lookback window*). We train GMMs to predict one-step-ahead, and then we use them recursively to obtain multiple steps ahead predictions. See, e.g., [16] for more details on how to train and use GMMs in the context of time-series forecast. We refer to this predictor as GMM.

Beside predictors *i)* and *ii)*, we can a use genie predictor, which knows the whole sequence of future arrivals and flow

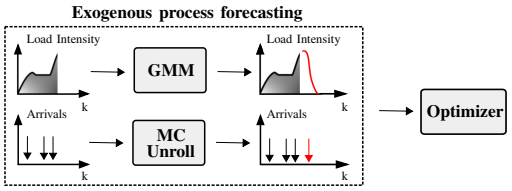


Fig. 2: **Exogenous process forecasting.** Arrival times and flow intensity forecasts are used by the MPC-based optimizer.

intensities, for all time slots k. We refer to the genie predictor with the letter G. Therefore, we can consider the following combination of the just described predictors GG, CG, GGMM, and CGMM. Our *goal* is to show that even using estimates about future PC arrivals can substantially improve the performance of VFI instantiation, chaining and scheduling, leading to an effective edge infrastructure control.

IV. Problem Formulation

Next, we formulate the edge control problem via Integer Linear Programming (ILP). First, we define the open-loop control problem for our system over a finite time horizon of T slots. Then, we close the control loop using a Model Predictive Control approach (receding horizon). We start by defining the edge server state.

Edge Server State: the state of an edge server $i \in \mathcal{N}^e$ is described by the number of VF instances of type m,

$$s_{i,m}(k+1) = s_{i,m}(k) + y_{i,m}(k) - \tilde{y}_{i,m}(k),\ \forall m \in \mathcal{M}. \quad (1)$$

State equation (1) is used to track the state evolution from the current time slot k to a future time slot $k+p$, with $p \in \{1,\dots,T-1\}$. Using (1), the evolution from slot k to slot $k+p$ is described as

$$s_{i,m}(k+p) = s_{i,m}(k) + \sum_{j=k}^{k+p-1} \left(y_{i,m}(j) - \tilde{y}_{i,m}(j)\right). \quad (2)$$

A. System Constraints

Here, we specify the system constraints over the time horizon, i.e., $k = 0, 1, \dots, T-1$.

VFI placement: constraint sets (3) bounds the number of VFIs that can be deployed on a given edge node $i \in \mathcal{N}^e$, i.e., they can never be negative or exceed V_i.

$$\begin{aligned} \sum_{m\in\mathcal{M}} s_{i,m}(k) &\leq V_i,\ i \in \mathcal{N}^e,\ m \in \mathcal{M}, \\ 0 \leq s_{i,m}(k) &\leq V_i,\ i \in \mathcal{N}^e,\ m \in \mathcal{M}, \end{aligned} \quad (3)$$

where state variable $s_{i,m}(k)$ is expressed according to (2).

Processing chain assignment: to ease the notation, we define the map $t(s_q, l) : \mathcal{S} \times \{1,\dots,L_{s_q}\} \to \mathcal{M}$, mapping stages $l \in 1,\dots,L_{s_q}$ of processing chain q onto the corresponding virtual function $m \in \mathcal{M}$. The following constraints (4) ensures that the total number of flows requiring type-m processing that are assigned to node i do not exceed the amount of VF processing capabilities of type-m at this node (number of tasks per time slot), i.e.,

$$\sum_{\substack{q\in\mathcal{Q}_k,\\ l\in\mathcal{N}_{s_q}^{Pc}: t(s_q,l)=m}} r_q x_i^{q,l} \leq R_m^{\text{vf}} s_{i,m}(k), \quad (4)$$

for $i \in \mathcal{N}^e$, $m \in \mathcal{M}$. The set of constraint (5) avoids partial assignment of processing chain stages to the sink node,

$$x_{\text{sink}}^{q,l} = x_{\text{sink}}^{q,l+1},\ l = 1,\dots,L_{s_q}-1. \quad (5)$$

VFIs chaining: set (6) ensures that demands of any virtual link $(u,v) \in \mathcal{E}_{s_q}^{Pc}$ assigned to physical link $(i,j) \in \mathcal{E}$ never exceed its capacity,

$$\sum_{q\in\mathcal{Q}_k} \sum_{(u,v)\in\mathcal{E}_{s_q}^{Pc}} r_q x_{ij,uv}^{q,l} \leq b_{ij},\ (i,j) \in \mathcal{E}. \quad (6)$$

Flow conservation constraints (7) ensure that packets are correctly routed between the assigned VF instances, without exceeding the link capacities. Indicating with $v_l = (l, l+1)$ the virtual link connecting stages l and $l+1$ of PC q, we have

$$\sum_{j\in\mathcal{N}} x_{ij,v_l}^q - \sum_{j\in\mathcal{N}} x_{ji,v_l}^q = x_i^{q,l} - x_i^{q,l+1}, \quad (7)$$

where $q \in \mathcal{Q}_k, i \in \mathcal{N},\ l = 1,\dots L_{s_q}-1$. Along with (7), we define flow conservation constraints for the first virtual link $v_{a_q} = (a_q, 1)$ connecting the generating access node $a_q \in \mathcal{N}^a$ to the node hosting the first processing stage for PC q,

$$\sum_{j\in\mathcal{N}} x_{ij,v_{a_q}}^q - \sum_{j\in\mathcal{N}} x_{ji,v_{a_q}}^q = \chi(i,a_q), \quad (8)$$

where $\chi(x,y)$ returns the value 1 if $x = y$, and 0 otherwise.

Load conservation: constraints (9) ensure that any PC stage is assigned either to an edge, the cloud or the sink node,

$$\sum_{i\in\mathcal{N}} x_i^{q,l} = 1,\ \forall q \in \mathcal{Q}_k,\ k = 0,1,\dots,T-1. \quad (9)$$

Quality of Service guarantees: moreover, we enforce end-to-end latency constraints to provide some guarantees on the latency experienced by the tasks flowing through processing chain $q \in \mathcal{Q}_k$.

$$\sum_{(i,j)\in\mathcal{E}} \sum_{(u,v)\in\mathcal{E}_{s_q}^{Pc}} x_{ij,uv}^q d_{ij} + \sum_{i\in\mathcal{N}} \sum_{l\in\mathcal{N}_{s_q}^{Pc}} x_i^{q,l} \tau_{i,t(s_q,l)}^{\text{vf}} \leq D_{s_q}. \quad (10)$$

Constraints (10) accumulate all the latency contributions along the selected path for PC q from its access node $a \in \mathcal{N}^a$ to its last processing stage, due to both transmissions and processing. With $\tau_{i,t(s_q,l)}^{vf}$ we indicate the execution time required by stage l of PC q assigned to node i that requires a type-m virtual function, with $m = t(s_q, l)$.

Integer variables: contraints (11) force decision variables to be integer,

$$y_{i,m}(k),\ \tilde{y}_{i,m}(k) \in \{0,1,\dots,V_i\},\ x_i^{q,l},\ x_{ij,uv}^q \in \{0,1\}. \quad (11)$$

Note that (11) makes the problem hard to solve.

B. Objective Functions

To optimize system operations, we need to properly define all the operating costs. We collect the decision variables in $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ collect PC assignment and chaining, and edge control decision variables, respectively. The overall costs for time slot $k = 0,1,\dots,T-1$ is

$$J(k) = J^{Pc}(k) + J^{\text{control}}(k), \quad (12)$$

where $J^{Pc}(k) = J^{\text{proc}}(k) + J^{\text{tx}}(k)$ accounts for PC related costs (processing and transmission), while $J^{\text{control}}(k) = J^{\text{idle}}(k) + J^{\text{d}}(k) + J^{\text{r}}(k)$ encodes the costs due to the management of the edge infrastructure (i.e., deployment, removal and idling of VF instances).

Edge control cost: collects the costs associated with the edge infrastructure management operations

$$J^{\text{ctrl}}(k) = \sum_{\substack{i\in\mathcal{N}^{\text{e}},\\ m\in\mathcal{M}}} \left(\xi^{\text{idle}}_{i,m}\cdot s_{i,m} + \xi^{\text{dep}}_{i,m}\cdot y_{i,m} + \xi^{\text{rmv}}_{i,m}\cdot \tilde{y}_{i,m}\right)(k). \tag{13}$$

The idling cost $J^{\text{idle}}(k)$ of VF instances is due to the reservation of resources on edge nodes to keep the considered VF active, i.e., the cost incurred in leasing memory space on edge devices. The VFI deployment and the VFI removal costs for time slot k are $J^{\text{d}}(k)$ and $J^{\text{r}}(k)$, respectively.

Processing chain execution and transmission costs: the processing cost incurred in the processing stages and in the transmission of the flow of PC $q \in \mathcal{Q}_k$,

$$J^{\text{Pc}}(k) = \sum_{q\in\mathcal{Q}_k}\left(\sum_{l\in\mathcal{N}^{Pc}_{s_q}} \xi^{\text{proc}}_{i,t(s_q,l)}\cdot x^{q,l}_i + \sum_{(u,v)\in\mathcal{E}^{Pc}_{s_q}} \xi^{\text{tx}}_{ij}\cdot x^{q}_{ij,uv}\right)\cdot \tag{14}$$

Note that, the assignment of a stage to the sink node is highly penalized ($\xi^{\text{proc}}_{\text{sink},m} \gg 1$) to discourage PCs rejection.

C. Finite Horizon Open-loop Control Problem

Next, we formulate the open-loop problem to be solved over a finite horizon of T time slots. Its solution returns control actions (edge VFI deployment/removal operations), processing chain assignments and chaining decisions.

$$\begin{aligned} &\underset{\boldsymbol{z}}{\text{minimize}} && \sum_{k=0}^{T-1} J(k) \\ &\text{subject to:} && (3)\text{–}(11). \end{aligned} \tag{15}$$

D. Predictive Control of Edge Computing Resources

To close the control loop, we use Model Predictive Control. According to this technique, we need *i)* to solve the open-loop problem with a prediction horizon $W_{\text{ph}} \leq T$ for each time step $k = 0, \ldots, T-1$. Usually, a $W_{\text{ph}} \ll T$ is selected. Note that, PC arrival process for time slots $k+1, \ldots, k+W_{\text{ph}}-1$ is unknown, and therefore it needs to be estimated (see Section III-E). *ii)* Apply the computed control and assignment actions just for the current time slot k. *iii)* Move to the next time slot $k+1$, and iterate these three steps for $k = 0, 1, \ldots, T-1$.

E. Heuristic Control of Edge Computing Resources

In what follows, we propose a heuristic edge control policy ($\text{HEU}(\epsilon_{\text{th}}, \epsilon_{\text{hy}}, \eta_{\text{avg}})$) that separates the VF instance placement phase from the assignment and chaining phases. The VFIs placement (first phase) is carried out as follows.

Phase 1: to decide whether to scale out/in type m virtual function instances or not, we propose the following algorithm.

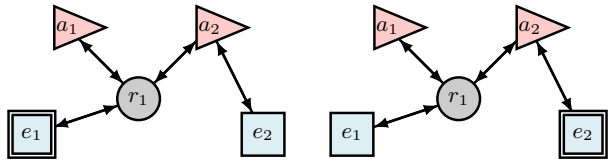


Fig. 3: **Heuristic control.** Where to place a new instance (double square) is decided based on *i)* type-m function generation frequencies $f_{a_j,m}$ for access nodes (triangles) a_j, $j = 1, 2$, and *ii)* distance between the selected edge node and the access nodes. When access nodes have similar generation frequencies $f_{a1,m} \simeq f_{a_2,m}$ select server e_1 (left), while in the case $f_{a_2,m} \gg f_{a_1,m}$ the server to be picked is e_2 (right).

Step 1: compute the blocking rate for the type-m virtual function. For this, a sliding window of the past η_{avg} time slots is used ($\eta_{\text{avg}} = 1$ indicates that the instantaneous blocking rate is considered at each slot k).

Step 2: the blocking rate for every type-m VF, computed in Step 1, is compared against a blocking rate threshold ϵ_{th} to decide whether to deploy a new m-type VFI, remove an existing one, or do nothing. Note that, a hysteresis ϵ_{hy} on the deployment/un-deployment cycle can be used.

Step 3: lastly, we need to compute where to deploy the new VFI, or which one of the already existing VFIs has to be removed by solving the scale-out $\boldsymbol{P_1^{\text{out}}}$ and the scale-in $\boldsymbol{P_2^{\text{in}}}$ problems, respectively,

$$\begin{aligned} \boldsymbol{P_1^{\text{out}}}) \quad i^*_{\text{s.out}} &= \underset{i\in\mathcal{N}^e\cap\mathcal{N}^{\text{s.out}}}{\text{argmin}} \sum_{a\in\mathcal{N}^a} (f_{a,m}(k)\cdot \texttt{dist}(i,a)) \\ \boldsymbol{P_2^{\text{in}}}) \quad i^*_{\text{s.in}} &= \underset{i\in\mathcal{N}^e\cap\mathcal{N}^{\text{s.in}}}{\text{argmax}} \sum_{a\in\mathcal{N}^a} (f_{a,m}(k)\cdot \texttt{dist}(i,a)), \end{aligned} \tag{16}$$

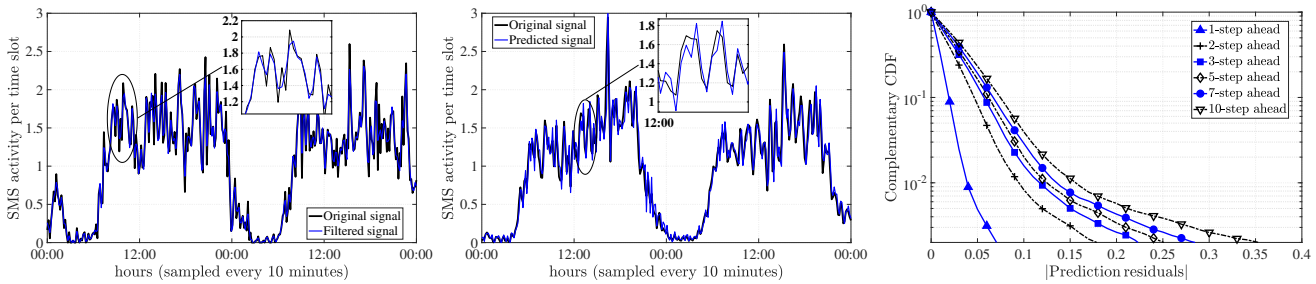
where $f_{a,m}(k)$ represents the average generation frequency of type-m processing stages at access node $a \in \mathcal{N}^a$, while $\texttt{dist}(i,j)$ is the distance between node i and j of graph G (on the shortest path). In Fig. 3, we show an example of how the generation frequency of access nodes $f_{a,m}(k)$ can affect deployment decisions. The two sets $\mathcal{N}^{\text{s.out}}$ and $\mathcal{N}^{\text{s.in}}$ collect those edge nodes that are respectively suitable for the deployment or removal of a type-m virtual function instance.

Phase 2: the heuristic assigns PC requests and chains VF instances jointly, by solving problem (15) without the edge control variables $y_{i,m}(k)$ and $\tilde{y}_{i,m}(k)$. Consequently, it heuristically optimizes the VFIs placement without considering the reconfiguration costs, and then optimally assigns processing chain requests to the available resources. In the predictive control case, instead, the placement, assignment, and chaining phases are solved jointly.

V. Numerical Results

A. Flow Intensity Dataset

We use the Open Big Data Challenge dataset collected by Telecom Italia from November 2013 to January 2014 [17]. It features mobile user activities (SMS, calls, and Internet use) over the city of Milan measured through Call Detail Records (CDRs), which are used for billing purposes and network management. This Dataset contains: *i) spatial aggregation*, activity measurements are provided for each square of a 100×100 grid, *ii) temporal aggregation*, activity measurements are obtained by temporally aggregating CDRs in time slots of 10 minutes.



(a) Original and low-pass filtered signals (b) Real and predicted signals (one-step ahead) (c) CCDF of residuals (multistep ahead)

Fig. 4: **Dataset and forecasts:** in *(a)*, the original and filtered signals are showed for two subsequent days. In *(b)*, an examples of real and predicted signals (1-step ahead GMM) are presented. In *(c)*, the complementary CDF of prediction residuals for $1, 2, 3, 5, 7$ and 10 time steps ahead are shown.

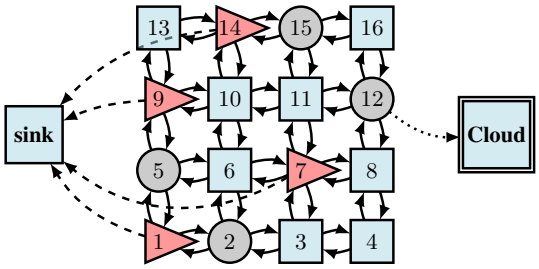


Fig. 5: **Test network example:** the substrate is composed of edge nodes (squares), access nodes generating traffic (triangles), and switch nodes forwarding traffic (circles). PCs can be offloaded to the cloud (double square), that is connected with the edge through a link with higher latency than the others (dotted arrows). Every access node is connected to the sink node through an artificial 0-latency link (dashed arrows).

We normalize traces to express the percentage of VF instances usage per time slot.

Original traces have been low-pass filtered (second-order Butterworth filter with a normalized cut-off frequency of 0.8), and then mapped onto a supervised learning problem, where inputs are sequences of $W_{\text{lb}} = 24$ consecutive past measurements $r_{a,s}(k - W_{lb} + 1), \ldots, r_{a,s}(k)$, and the target variable is $r_{a,s}(k + 1)$, $k = 0, 1, \ldots$, (one-step-ahead prediction). As anticipated in Section III-E, we use GMMs to predict flow intensity processes $r_{a,s}(k)$, although the approach is general and other forecasting techniques may be used. In Fig. 4b, a qualitative evaluation of the GMM prediction accuracy over a time span of two days is shown. Recalling that we are interested in multistep-ahead forecasting, i.e., measurement $k+1,\ldots, k+W_{\text{ph}}$, we use one-step ahead GMMs recursively to obtain the desired number of future forecast values. Although this approach accumulates error as the number of steps ahead increases, it allows us to train a single GMM model per access node and processing chain type. Fig. 4c quantifies how much the prediction performance degrades as the number of prediction steps ahead increases. The metric shown is the Complementary Cumulative Distribution Function (Complementary-CDF) of the prediction residuals $|r_{a,s}(k) - \hat{r}_{a,s}(k)|$, where variables $\hat{r}_{a,s}(k)$ represent the predictions at time k.

B. Definition of Performance Metrics

To assess the performance of the two approaches, i.e., MPC-based and heuristic, in sections IV-E and IV-D, respectively, we consider the following metrics.

i) The ***PC requests blocking rate*** P_{BR}, expressed as the average rate of processing chain requests that the system is not able to satisfy (i.e., that are assigned to the sink node):

$$P_{\text{BR}} = \frac{1}{T}\sum_{k=0}^{T-1}\sum_{q\in\mathcal{Q}_k}\sum_{l\in\mathcal{N}_{s_q}^{Pc}}\frac{x_{\text{sink}}^{q,l}}{L_{s_q}}, \tag{17}$$

where $x_{sink}^{q,l} = 1$ indicates an assignment of the l-th stage of PC q to the sink node. Similarly, we can compute edge and cloud processing rates for PC stages P_{ER} and P_{CR}, respectively.

ii) The ***overhead cost*** C_{OV} counts the average number of control actions across the edge infrastructure (VF instances deployments and removals) per time slot

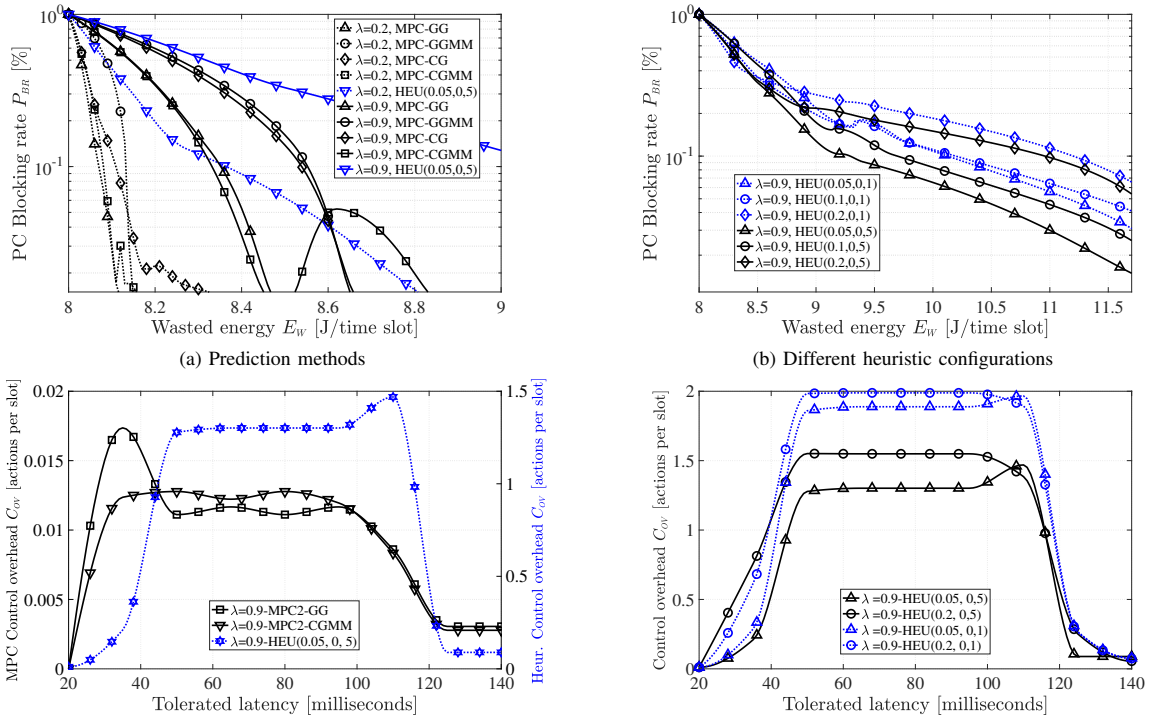
$$C_{\text{OV}} = \frac{1}{T}\sum_{k=0}^{T-1}\sum_{i\in\mathcal{N}^e}\sum_{m\in\mathcal{M}}(y_{i,m}(k) + \tilde{y}_{i,m}(k)). \tag{18}$$

iii) The last metric quantifies the so called ***wasted energy cost*** E_W. Let us define the cumulative amount of idling VFI resources of type m for time slot k on edge node i as $u_{i,m}^{\text{idle}}(k) = R_{i,m}^{\text{vf}} s_{i,m}(k) - \sum_{q\in\mathcal{Q}_k}\sum_{l\in\mathcal{N}_{s_q}^{Pc}:t(s_q,l)=m} r_q x_i^{q,l}$, and the cumulative amount of Mb transmitted over the link $(i,j) \in \mathcal{E}$ during time step k as $u_{ij}^{\text{tx}}(k) = \sum_{q\in\mathcal{Q}_k}\sum_{(u,v)\in\mathcal{E}_{s_q}^{Pc}} r_q x_{ij,uv}^{q}$. Then, the wasted energy cost is defined as $E_W = E_{\text{IDLE}} + \alpha E_{\text{TX}}$, where E_{IDLE} represents the cost of idling VFI resources per time slot, E_{TX} is the transmission cost, and α is a weighting factor that is used to prioritize one cost over the other.

$$\begin{aligned} E_{\text{IDLE}} &= \frac{1}{T}\sum_{k=0}^{T-1}\sum_{i\in\mathcal{N}^e}\sum_{m\in\mathcal{M}}\xi_{i,m}^{\text{idle}} u_{i,m}^{\text{idle}}(k), \\ E_{\text{TX}} &= \frac{1}{T}\sum_{k=0}^{T-1}\sum_{(i,j)\in\mathcal{E}}\xi_{ij}^{\text{tx}} u_{ij}^{\text{tx}}(k). \end{aligned} \tag{19}$$

C. Network Scenario

The test network has $|\mathcal{N}^e| = 8$ edge computing-enabled devices, $|\mathcal{N}^a| = 4$ access nodes (injecting traffic), $|\mathcal{N}^r| = 4$ network switches, $|\mathcal{N}^c| = 1$ cloud premises, and an artificial sink node, see Fig. 5. Node positions within the grid are randomly assigned for each generated network. A single link connects the edge region (grid) to the cloud, and every access node is connected to the artificial sink (to reject processing chains that cannot be executed due to insufficient resources). Since the edge is composed of resource-limited devices



(a) Prediction methods

(b) Different heuristic configurations

(c) Control overhead (scales are different)

(d) Control overhead of heuristic configurations

Fig. 6: **Control overhead and PC blocking rate:** in *(a)* MPC performance ($W = 2$) is shown for increasingly relaxed latency requirements (from unfeasible requirements (top-left corner) to cloud-feasible req.), for a low congested scenario $\lambda = 0.2$ (dotted curve), and a highly congested one $\lambda = 0.9$ (solid curve). Moreover, the performance obtained with the heuristic approach $\mathrm{HEU}(0.05, 0, 5)$ is reported. In *(b)*, heuristic performance for different parameters configuration are outlined. In *(c)*, the control overhead required by the MPC-based and the heuristic approaches are compared (note that axes have different scales). In *(d)* the control overhead curves produced by different configurations of the heuristic method are compared.

($V_i = 1$ parallel VFIs), the controller has to route multistage services through it, according to both the transmitting and computing resources that are available, while ensuring that QoS requirements are met. We consider processing chains requests of type $s \in \mathcal{S}$, $|\mathcal{S}| = 1$, composed of two stages. Each stage can request one of the two virtual functions available in $\mathcal{M}$ ($|\mathcal{M}| = 2$). Edge link latencies are drawn uniformly at random within the range $[2, 10]$ milliseconds, while the link connecting the edge to the cloud has an average latency of 100 milliseconds. Edge processing costs $\xi^{\mathrm{proc}}_{i,m}$, $i \in \mathcal{N}^e$ are assumed to be 0, i.e., the operator does not consider processing cost when exploiting deployed virtual functions at the edge (the provider already pays to keep VFI active and to deploy them). The cloud processing cost has been fixed to $\xi^{\mathrm{proc}}_{i,m} = 1, i \in \mathcal{N}^c$. Idling, deployment, and removal costs of VFI have been set to 1 ($\xi^{\mathrm{idle}}_{i,m} = \xi^{d}_{i,m} = \xi^{r}_{i,m} = 1$). We tested two processing chain arrival rates (of the Markov Chain) $\lambda_a = 0.2$ (low demand) and $\lambda_a = 0.9$ (congested), and we assume that all access nodes have the same arrival rate. Recall that λ_a represents the average number of arrivals per time slot associated with the Markov Chain describing the PC arrival process for an access node a (see Section III-D). Instead, flow intensity traces differ from access node to access node, i.e., we select different dataset radio cells. The task processing time $\tau^{\mathrm{vf}}_{i,m}$ is set to 10 milliseconds for all virtual functions $m \in \mathcal{M}$, and VF instances capacity is normalized to $R^{\mathrm{vf}}_{i,m} = 1$. For each generated network, we simulate $T = 720$ time steps (5 days considering a 10-minute sampling interval, i.e., $\Delta k = 10$ minutes) and we average the metrics over 12 randomly generated networks. The remaining simulation parameters are summarized in Table II.

TABLE II: Selected simulation parameters

Name	Value	Unit (for time slot Δk)
$\xi^{\mathrm{idle}}_{i,m}, \xi^{\mathrm{dep}}_{ij}, \xi^{\mathrm{rmv}}_{i,m}$	1	J per {actv|dep|rmv} VF instance
$\xi^{\mathrm{proc}}_{i,m}$	0 (1)	J per unit of proc. res. at Edge (Cloud)
ξ^{tx}_{ij}	1	J per unit of tx resource

D. Numerical Results

Next, we present three performance assessments: *1)* PC blocking rate P_{BR} *vs.* wasted energy cost E_W, *2)* control overhead C_{OV} *vs.* QoS requirements, and *3)* PC blocking rate P_{BR} *vs.* QoS requirements. For all the curves, the prediction horizon of the MPC-based approach is set to $W_{\mathrm{ph}} = 2$ slots. **Case 1:** we investigate how the optimal (ILP) and heuristic allocation methods behave when using different configurations on the (P_{BR}, E_W)-plane, see Figs. 6a and 6b. These curves have been obtained by simulating the system for increasing values of the tolerated latency (most latency-sensitive requirements lie in the top-left corner). In Fig. 6a, we address how the 4 different prediction methods described in Section III-E affect the performance of the MPC-based algorithm, for low and high arrival rates $\lambda_a = 0.2$ and $\lambda_a = 0.9$, respectively. As expected, the predictive approach always overcomes the heuristic one. MPC using non-ideal predictors for both arrival instants $\delta_{a,s}(k)$ and flow intensity $r_{a,s}(k)$, termed CGMM,

exhibits oscillations in the blocking rate. A predictive control policy, exploiting information about future demands, can reduce the blocking rate of PC requests by 95% with respect to the heuristic policy while maintaining the wasted energy at a similar level. In Fig. 6b, instead, we compare the performance of heuristic method HEU$(\epsilon_{\text{th}}, \epsilon_{\text{hy}}, \eta_{\text{avg}})$ with several combinations of its parameters $\epsilon_{\text{th}}, \epsilon_{\text{hy}}$, and η_{avg}. We note that an average window (on the blocking rate) $\eta_{\text{avg}} = 1$ performs worse than $\eta_{\text{avg}} = 5$. For example, the heuristic policy HEU$(0.05, 0, 5)$ achieves a blocking rate of 0.1 wasting 9% and 16% less energy than HEU$(0.05, 0, 1)$ and HEU$(0.2, 0, 5)$, respectively. This result means that smoothing high-frequency oscillations on the blocking probability signal (through a larger η_{avg}) helps the heuristic approach reduce its wasted energy and that the blocking probability threshold ϵ_{th} highly affects heuristic policy performance. Note that smaller values of the threshold ϵ_{th} lead to a reduction of the PC blocking rate P_{BR}. **Case 2:** in Figs. 6c and 6d we assess the control overhead induced on the edge infrastructure while varying the tolerated latency of the processing chain requests from 20 to 140 milliseconds (there is at least an average delay of 100 milliseconds to reach the cloud). In Fig. 6c, the control overhead C_{OV} required by the MPC-based and the heuristic approaches are compared. Note that the scale of the two axes is different, and the MPC-based method induces a much lower control overhead. At a tolerated latency of 30 and 70 milliseconds, the predictive approach respectively leads to 87% and 99% less control overhead than the heuristic one, HUE$(0.05, 0, 5)$. In Fig. 6d, multiple C_{OV} curves obtained using different configurations of the proposed heuristic are compared. We observe that an average window $\eta_{\text{avg}} = 5$ induces a smaller overhead (up to 31%) than $\eta_{\text{avg}} = 1$. This fact is due to the higher frequency oscillations in the blocking rate signal used to decide when reconfiguring the VFIs deployed on the edge infrastructure, which are compensated for by a larger window. **Case 3:** lastly, we report the processing chain blocking rates P_{BR} as the tolerated latency by PC's tasks increases. In Fig. 7, MPC-based curves with a prediction horizon $W_{\text{ph}} = 2$ and prediction methods GG (Genie predictors for both arrival and flow intensities) and CG (MC unrolling arrivals and Genie flow intensity predictors) are shown. These are the best and the worst performing MPC-based schemes. Moreover, we display P_{BR} curves for some selected heuristic configurations. The MPC-based approach can better control the edge infrastructure allowing the reduction of the PC blocking rate (by 93% at a tolerated latency of 50 milliseconds) while wasting fewer processing and transmission resources.

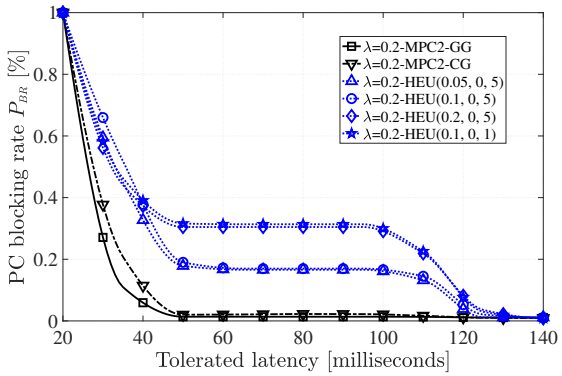


Fig. 7: **Results case 3:** PC blocking rates are evaluated for increasing values of the tolerated latency and for different heuristic configurations.

VI. Conclusions

The predictive control approach for virtual function placement, assignment and chaining that has been presented and evaluated in this paper confirms the effectiveness of exploiting prediction of exogenous processes (computing demand) into the allocation of edge computing resources in edge networks. Despite the excellent performance, one limitation of such an approach is its complexity (ILP problem). To effectively scale up this allocation technique, work still has to be performed on suitable relaxations to efficiently tackle the ILP formulation, while retaining the advantages of the optimal solver.

References

[1] K. Razavi, L. M. Razorea, and T. Kielmann, "Reducing VM Startup Time and Storage Costs by VM Image Content Consolidation," in *Euro-Par 2013: Parallel Processing Workshops*, Berlin, Heidelberg, 2014.

[2] F. Esposito, D. Di Paola, and I. Matta, "On Distributed Virtual Network Embedding With Guarantees," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, 2016.

[3] Y. Zhu and M. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *IEEE INFOCOM*, Barcelona, Spain, 2006.

[4] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, 2008.

[5] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015.

[7] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Computer Communications*, vol. 102, 2017.

[8] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *International Conference on Network and Service Management (CNSM)*, 2014.

[9] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal Dynamic Cloud Network Control," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, 2018.

[10] C. Wang, J. Llorca, A. M. Tulino, and T. Javidi, "Dynamic Cloud Network Control Under Reconfiguration Delay and Cost," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, 2019.

[11] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, 2018.

[12] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *IEEE 4th International Conference on Cloud Networking (CloudNet)*, Niagara Falls, Canada, 2015.

[13] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction," in *IEEE INFOCOM*, Honolulu, HI, 2018.

[14] R. Li, Z. Zhou, X. Chen, and Q. Ling, "Resource Price-Aware Offloading for Edge-Cloud Collaboration: A Two-Timescale Online Control Approach," *IEEE Transactions on Cloud Computing*, 2019.

[15] M. Gaggero and L. Caviglione, "Predictive Control for Energy-Aware Consolidation in Cloud Datacenters," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 2, 2016.

[16] M. Scalabrin, M. Gadaleta, R. Bonetto, and M. Rossi, "A Bayesian forecasting and anomaly detection framework for vehicular monitoring networks," in *IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2017.

[17] Telecom Italia Mobile, "Open big data challenge." [Online]. Available: https://dandelion.eu/datamine/open-big-data/