# Optimized Sampling Strategies to Model the Performance of Virtualized Network Functions

Steven van Rossem, Wouter Tavernier, Didier Colle, Mario Pickavet, Piet Demeester

Cover Page
Optimized Sampling Strategies to Model the Performance of Virtualized Network Functions

**Corresponding author**
Steven Van Rossem
Albert Lienartstraat 13, B-9300 Aalst, Belgium
stevenvanrossem@gmail.com
+32485750365

**Other authors**
Wouter Tavernier
Didier Colle
Mario Pickavet
Piet Demeester
Ghent University - imec, IDLab
iGent Tower - Department of Information Technology
Technologiepark-Zwijnaarde 126, B-9052 Ghent, Belgium
{firstname}.{surname}@ugent.be
+32 9 33 14920

**Noname manuscript No.**
(will be inserted by the editor)

# Optimized Sampling Strategies to Model the Performance of Virtualized Network Functions

**Steven Van Rossem · Wouter Tavernier · Didier Colle · Mario Pickavet · Piet Demeester**

Received: date / Accepted: date

**Abstract** Modern network services make increasing use of virtualized compute and network resources. This is enabled by the growing availability of softwarized network functions, which take on major roles in the total traffic flow (such as caching, routing or as firewall). To ensure reliable operation of its services, the service provider needs a good understanding of the performance of the deployed softwarized network functions. Ideally, the service performance should be predictable, given a certain input workload and a set of allocated (virtualized) resources (such as vCPUs and bandwidth). This helps to estimate more accurately how much resources are needed to operate the service within its performance specifications. To predict its performance, the network function should be profiled in the whole range of possible input workloads and resource configurations. However, this input can span a large space of multiple parameters and many combinations to test, resulting in an expensive and overextended measurement period. To mitigate this, we present a profiling framework and a sampling heuristic to help select both workload and resource configurations to test. Additionally, we compare several machine-learning based methods for the best prediction accuracy, in combination with the sampling heuristic. As a result, we obtain a reduced dataset which can still model the performance of the network functions with adequate accuracy, while requiring less profiling time. Compared to uniform sampling, our tests show that the heuristic achieves the same modeling accuracy with up to five times less samples.

**Keywords** Sampling Heuristic · Network Function Virtualization · Performance Profiling · Machine Learning · Regression

## 1 Introduction

In the telecom industry, there is an increasing adoption of cloud-native services and network functions based on Software Defined Networking (SDN) and Network Function Virtualization (NFV) techniques. By virtualizing compute and network resources, a very flexible environment can be created to deploy Virtual Network Functions (VNFs) with an optimal amount of allocated resources, adapted to the realtime incoming workload. The recent rise of 5G enabled services further advocates the use of cloud-native functions, which are deployed

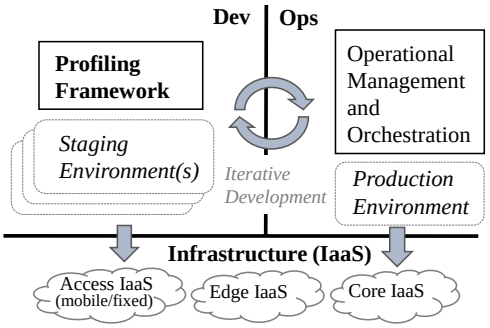



**Fig. 1** The profiling framework can use a similar infrastructure compared to operations, as part of a DevOps workflow.

This work has been performed in the framework of the NG-PaaS and 5GTANGO project, funded by the European Commission under the Horizon 2020 and 5G-PPP Phase2 programmes, resp. under Grant Agreement No. 761 557 and 761 493 (http://ngpaas.eu) (https://www.5gtango.eu). This work is partly funded by UGent BOF/GOA project "Autonomic Networked Multimedia Systems".

All authors are at
Ghent University - imec, IDLab
E-mail: {firstname}.{surname}@ugent.be

The corresponding author is:
Steven Van Rossem E-mail: stevenvanrossem@gmail.com

over a virtualized infrastructure [1] [2]. This illustrates the growing need to map the amount of allocated resources and incoming workload to the Key Performance Indicators (KPIs) of the deployed network service, specified in the Service Level Agreement (SLA). To characterize this relation as good as possible, we propose that the virtualized function is profiled on the targeted infrastructure, before being deployed in production. This is illustrated in Fig. 1 as a DevOps inspired workflow. Cloud-native techniques enable a very flexible use of the Infrastructure as a Service (IaaS) for a wide variety of use cases, as envisioned in [3]. This enables the profiling framework (Dev) to do its testing on a representative (but isolated) part of the IaaS, compared to the operational environment (Ops).

In this paper, we propose an optimized sampling procedure to shorten the profiling time as much as possible, without losing modeling accuracy. This optimized profiling procedure greatly benefits a DevOps-inspired deployment cycle: frequent VNF updates can be quickly and representatively validated, before the updated VNF is handed over to the service provider for deployment in production. We hereby consider the VNF as a black-box, with no formal way to deterministically calculate the performance metrics. VNF Profiling is then basically a form of load testing, where representative workloads are emulated and performance is recorded. The data analysis investigated in [4], shows further that the output data of the profiling tests can be poured into a model; such a model can predict from the resource allocation and expected workload, the performance level of the profiled VNFs. The validation of this model in the profiling environment, brings additional trust when deploying the profiled VNF in the production environment. The research goal of this article is to optimize the data gathering part of the profiling procedure as much as possible. This is done by investigating whether the total test space of possible workloads and resource allocations can be limited, without losing too much prediction accuracy. We compare several machine-learning based methods for the best performance modeling accuracy, in combination with several sampling heuristics.

But first, in Section 2, we compare our work to state-of-the-art approaches in VNF performance modeling. Next, in Section 3, we present the architecture and implementation of our automated profiling tool, which we used to gather the profiling data. In Section 4 we present the VNFs used in our experiments. Then in Section 5, the general performance trends we witnessed on the tested VNFs are analysed. Finally, Section 6 uses the profiled data to evaluate several machine-learning-based methods for the best performance modeling accuracy, in combination with multiple sampling heuristics.

## 2 Related Work

In this paper we unite two different domains: On one hand, platforms for automated VNF testing and on the other hand, multi-variate sampling to model a certain response function as efficiently as possible. Efficiency in the context of this paper means as little samples as needed, to obtain an accurate model of the VNF performance, in the shortest time frame possible. In Table 1, we give an overview of the related research in these two domains. Our presented method builds further upon this previous work, and achieves a higher sampling efficiency. The problem of efficient performance sampling is not unique for the NFV domain, related work can be found in other application fields, as indicated in the table. For each referenced related work, we shortly describe if a sampling heuristic was used or not, and any drawbacks we see, compared to our method.

Various base principles for profiling virtual environments were first discussed in [5]. Some the main conclusions in this reference also apply to our use case: Profiling tests must be able to apply a range of representative workload intensities and should run in nearly-identical environments compared to production. Additionally, linear regression is exemplified to be a good modelling method for CPU usage. Further work is however needed to expand the methodology in [5] to non-linear regions, where resources such as CPU get saturated. It is also possible to optimize the sampling procedure, by proactively defining significant workloads to test.

Every reference in the NFV domain in Table 1, makes use of a platform to automate VNF performance measurements. Generic architectures for profiling frameworks have been described earlier in [6, 7, 8, 9, 11, 12], where also the relation to DevOps related workflows is highlighted. We extend this previous work with more insights for using a Service Oriented Architecture (SOA) and integration of both sampling and modeling methods. Also we exemplify VNF profiling in a more elaborate parameter space of both workload and resource metrics.

The VNF profiling platforms in the first two rows of Table 1 exhaustively executes a list of input benchmark tests, without trying to optimize the test time. Several sampling strategies for VNF performance profiling have been studied before and combined with automated profiling, in the last seven rows of Table 1:

In [13], it is experimentally observed that VNF performance shows a trend break when resources are getting saturated. During profiling, this approach increases the workload in fixed steps until resource saturation occurs. We argue that our approach, based on bisection, provides a faster way to model the VNF performance trend, needing less samples (as we will see in Section 5.1).

| Reference | Domain | Sampling Efficiency | Sampling heuristic | Drawbacks |
|---|---|---|---|---|
| Wood [5] | Cloud computing | - | Exhaustive sampling using CPU, network and disk intensive microbenchmarks. Post process the training set to filter anomalous measurements. | Not time-efficient, covers only part of the operational space. |
| z-Torch [6]<br>NFV-vital [7]<br>NFV-inspector [8] | NFV | + | Manually selected parameters and values | Covers only part of the operational space. |
| SONATA SDK [9, 10]<br>Gym [11]<br>Probius [12] | NFV | - | Exhaustive sampling | Not time-efficient, not scalable. |
| ORCA [13] | NFV | + | Stepwise increase workload until resource saturation + exhaustive resource combinations. | More time-efficient but not scalable. |
| Duplyakin [14] | Cloud computing | ++ | Multi- variate Gaussian Process to select the next samples. | Needs many initial samples, less scalable. |
| GEIST [15] | Cloud computing | + | Random uniform in selected areas via CAMLP (label propagation). | Not fit for trend modelling since only global optimum finding. |
| Sumo [16] | Generic | ++ | Voronoi partitioning of the parameter space + KPI gradient-based selection of interesting partitions. | Very generic, needs large pool of initial uniform samples. |
| PANIC [17] | Cloud computing | ++ | Greedy, KPI gradient- based bisection on all parameters. | Less scalable. |
| Peuster [18] | NFV | +++ | Set parameter weights via KPI gradient + random sampling on weighted parameters. Fixed workload over multiple resource combinations. | Less scalable to many workload parameters, can outperform PANIC but marginally outperforms random uniform sampling. |
| Giannakopoulos [19] | Cloud computing | +++ | Use piecewise linear functions to model the KPI trends, bisect areas with highest linearity deviation. | Can outperform PANIC but marginally outperforms random uniform sampling. |
| This paper | NFV | ++++ | Scalable PLS-based parameter selection (phase1) + KPI gradient-based bisection (phase2) on selected workload metric + curve fitting accuracy as sampling stop criterium. Outperforms uniform sampling. | |

**Table 1** Related VNF perfomance sampling approaches and drawbacks.

The authors in [14] recommend the use of Gaussian Processes (GP) as a flexible method to both model performance and choose the next configurations to sample. Our tests show however that the trends witnessed in VNF profiling are not efficiently captured by GPs. Also when we let the GP model determine the next samples, the method remains sub-optimal compared to generic uniform sampling (see Section 6.2). Likewise, in [15], it is experimentally tested that uniform sampling and Gaussian Processes are not the best methods to sample the large parameter space of compute intensive algorithms. The used method is however aimed at finding a global optimum, which is not applicable to our use case, which tries to model the complete response surface.

Another commonly used adaptive sampling method is based on surrogate modeling using a gradient based approach [16]. This is a very generic method which can be applied in many domains, but the risk is that sampling efforts focus too hard on local extrema, and therefore take an unbalanced number of samples in only limited regions. Also a relatively large number of initial random samples is needed to feed the sampling algorithm.

The use of gradient based sampling selection is applied to VNFs in [17]. By using a greedy algorithm, this method becomes less scalable when multiple workload and resource parameters must be profiled, since a many possible combinations remain to be sampled.

Several sampling strategies for VNF chain profiling have been investigated in [18], but no method is found which significantly beats a generic uniform sampling strategy. Decision tree based models are put forward as a promising solution, however our tests show that random forest is one of the less performing methods

in our use-case of profiling a single VNF. The profiling tests in [18] are also limited to varying only one resource metric (allocated vCPU), under only one fixed workload.

The profiling method in [19] focuses on the deployment space of big data applications, with up to seven configuration dimensions on a fixed server, thus with a fixed set of resource parameters. In this deployment space, areas are clustered in which the performance metric can be approximated using a linear model. The total model is than a piecewise combination of different linear models and the space partitioning into different linear regions is done by a decision tree. More samples are adaptively chosen in the areas where a linear model has bad accuracy. However, when trying to model non-linear functions, a deteriorating performance is reported. Moreover, new sample points are drawn randomly in a uniform manner, without exploiting any expert knowledge. Our sampling heuristic tries to mitigate this by using a curve fit model with piecewise (non-linear) functions and online sample selection using bisection.

The scalability mentioned in Table 1, relates to how well the sampling heuristic can cope with additional parameters to test. Exhaustive or greedy methods need a lot of time to test the complete range of specified parameters, similar to methods which require a large number of initial samples. This is mitigated by including some form of feature selection. Our presented sampling procedure further augments the mentioned related work, with the PLS method [20], to select any significant workload or resource parameters to prioritize sampling on.

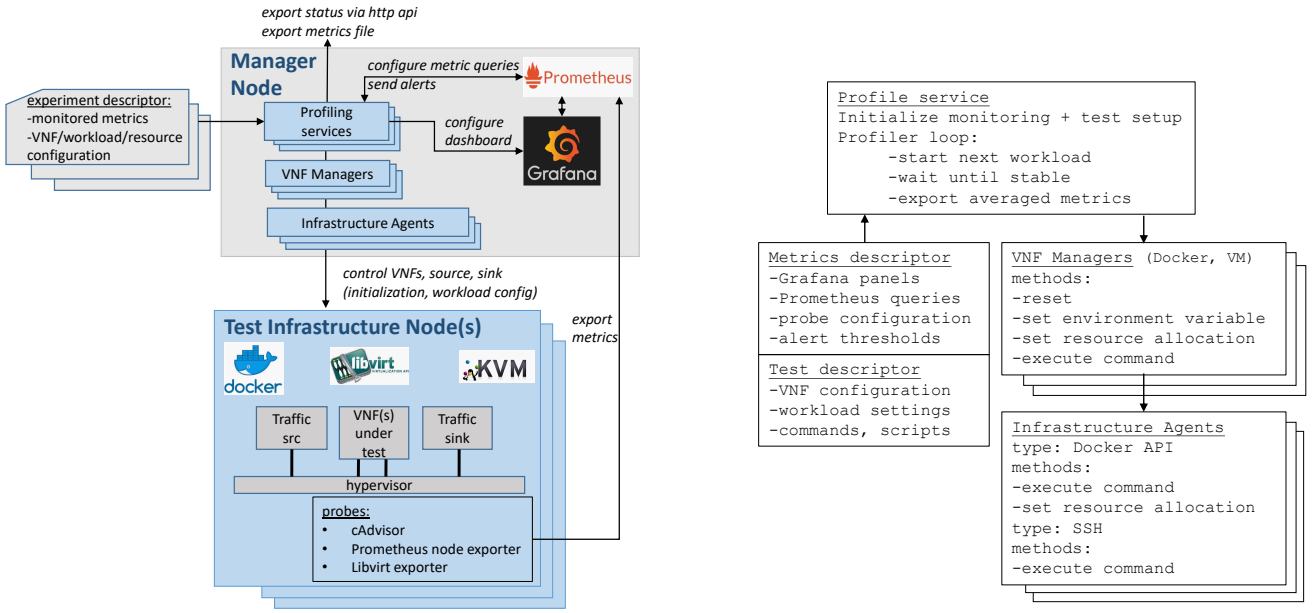
A reference architecture for VNF Management and Orchestration is called "ETSI MANO" [21], a standard maintained by the ETSI NFV working group. The ETSI MANO architecture is followed in our framework, in the sense that a strict separation between infrastructure and VNF management functionality is enabled. Also, management functions per VNF and per executed profiling test are integrated. According to ETSI MANO specifications, a VNF Manager (VNFM) is responsible for VNF lifecycle management (e.g. instantiation, update, query, scaling, termination), with a clear link to the infrastructure for updating resource allocations. Other VNF management related actions (such as VNF login or functional configuration) is done via the ETSI-defined Element Manager (EM). The VNF Manager entity in our setup (as will be explained in Section 3) incorporates characteristics of both the ETSI EM and VNFM. The main interface protocol used by the VNF Manager in our setup is either SSH or the Docker API. The EM in our setup can then be considered as the SSH or Docker agent instantiated in the infrastructure node where the VNF under test is running, addressed by the VNF Manager.

Our platform implementation for automated profiling is based on the development in [10] (which is also based on ETSI MANO). We re-factored the tool following a Service Oriented Architecture (SOA) approach, where multiple profiling tests can run independently and in parallel. As such, the tool can shorten the total profiling time, by parallelizing multiple measurement campaigns over multiple hardware nodes. In [9], a descriptor format is presented as input for an automated VNF profiling procedure. We have loosely adopted the syntax of this YAML-based test definition and included some adaptations to map it better to our envisioned sampling heuristic. The needed adaptations were mainly specialized configuration directives and settings for test execution and monitoring. In the next section we will explain our VNF profiling platform more in detail.

## 3 VNF Profiling Framework

The automation of VNF performance profiling is the main objective of our VNF profiling platform. In this section we first explain how such test automation was implemented. To further optimize the profiling procedure and save time, a sampling heuristic is further investigated in the next sections of this paper. We can put forward three main characteristics for a VNF profiling framework:

- **A light-weight, modular architecture** which is easily expandable. Since every VNF can have unique control interfaces or other configuration mechanisms, customization is necessary. The framework should therefore allow easy integration of custom VNF configuration functionality. Also parallel execution of profiling tests is an important feature to optimize profiling time. Details are given in Subsection 3.1.
- **Easy generation and reproducibility** of VNF profiling tests can be achieved by using descriptor files which contain a programmable testing and monitoring workflow. Such descriptor files allow an easily customizable profiling workflow per VNF. Details are given in Subsection 3.2.
- The integration of existing **cloud-native functional blocks** allows a fast development and flexible deployment on available IaaS. This cloud-native nature allows the profiling framework to be be quickly set up. As a result, we can test and measure VNFs on available IaaS nodes as a realistic staging environment (as seen in Fig. 1 and described in [3]). Details are given in Subsection 3.3.




(a) Architecture and used tools.

(b) Structural implementation of a profiling service.

**Fig. 2** Architecture and implementation of the profiling framework.

### 3.1 Modular Architecture

The purpose of VNF profiling is to generate a dataset which can be used to create a model for the VNF performance. The framework therefore automates a set of measurements where varying resources are allocated, workload is generated and KPIs are measured. Each test setup consists of a VNF under test, where the test traffic is routed from a traffic source to a traffic sink. The traffic source and sink can be considered as VNFs also, custom built and deployed for our test purpose. We hereby give a concise overview of this framework. The architectural overview is given in Fig. 2a. On the upper level, the Manager Node is where the profiling workflow is executed in a loop: (i) A workload and/or resource allocation is selected to test, (ii) instructions are given to the VNFs and traffic generators and (iii) monitored metrics are analyzed. On the lower level, the Test Infrastructure Nodes are located in the IaaS. This is the actual execution environment of the VNFs and traffic generators.

The **Manager Node** is where each profiling test is running as a separate service instance. By executing a profiling test as a separate service, multiple tests can be running simultaneously. This allows parallelization of multiple measurements. Each profiling test can be considered as a micro-service, from which the test status can be queried through an HTTP API. At the end of the profiling loop, the measurements are stored in .csv format for further analysis (See Section 5). A more detailed implementation diagram is depicted in Fig. 2b. We can distinguish following important class objects, addressed by the profiling service:

- Every VNF in the test setup (source, sink and VNF under test) has its own *VNF Manager* instance, which has methods available to control the VNF state. The most important function is to execute commands which start/stop traffic workloads. The actual commands or scripts are specified in the descriptor files, the VNF Manager only executes the defined command using the correct Infrastructure Agent. We have implemented a separate VNF Manager class for Docker containers and Virtual Machines (VMs), because they need different functions regarding (re)starting, or configuration.
- Every VNF Manager, should have an *Infrastructure Agent* attached, to address a specific API in the remote Infrastructure Test Node. This agent is used to execute commands inside a container or set container resources via the Docker API of the remote node or execute commands through SSH. Also specific VM hypervisors such as KVM can be addressed via an Infrastructure Agent.

Whenever a new type of VNF or infrastructure interface needs to be addressed, a new class type can be added in this framework. In this way, the profiling framework can be easily expanded or customized.

### 3.2 Profiling Descriptors

Profiling descriptors are YAML based documents, which describe the execution of a profiling test. They enable a programmable workflow which can be easily customized. We distinguish two main descriptor categories:

- Every profiling test is defined by a *Test descriptor*. Here we describe which scripts or commands to execute in each VNF, in order to generate the requested workload. See the example Listings 1 and 2.
- A *Metrics descriptor* defines for each VNF in the Test descriptor which metrics should be monitored and recorded. Care should be taken that all required metrics are exported by the pre-deployed probes. Additionally, it can be specified when to send alerts back to the profiling service when measurement stability or overload of the traffic src/sink is detected. See the example in Listing 3.

To have a more practical idea of the profiling execution, we give a succinct overview of the format used to describe the profiling tests. A first part of the *Test descriptor* is given above in Listing 1. This part defines a configuration agent for each VNF in the profiling setup. As previously explained, a class instance is made for each specified VNF manager and each manager uses an infrastructure agent as interface to the underlying VNF. The names of the VNF managers defined here, are referred to in the remainder of the descriptor. The declarations in the descriptor hold test-specific settings. For the agents this includes: API endpoints, credentials, authentication methods, ... For the VNF managers we need specific settings such as: the infrastructure agent to communicate with the VNF, container or VM uid, resource or operational initialization to be configured via the infrastructure agent, ...

```
agents:
  docker1:
    class: DockerApiClient
    url: 'tcp://docker.api.url:port'
  # ...
  ssh1:
    class: SshClient
    host: 'infrastructure.node.url'

managers:
  src:
    class: Docker
    agent: docker1
    cpuset_cpus: '8-15'
  # ...
  pfsense:
    class: Vm
    agent: ssh1
```

**Listing 1** YAML based test descriptor - interface configuration

A second part of the *Test descriptor* is then given in Listing 2. This is the most important part from the profiling perspective, as it defines the actual values for each relevant parameter in the test. For each parameter we define:

- A list or range of values to test (Line 4,10,14,25).
- The method of the manager instance to call, in order to practically configure this value into the VNF test (Line 5,11,15,26). In our tests, we use a common technique based on environment variables. The workload settings are stored in environment variables in the VNFs, later when the workload script is called, these variables are read and the configured workload is started.
- A list of fixed initialization commands, which are executed every time a new setting is configured (e.g. to stop/start a workload generating script in the traffic generator) (Line 18,29).

```
1  resource_parameters:
2      - name: pfsense_cpu_limit
3        # allocated cpu in %
4        values: [25,50,75,100,200,300,400,500]
5        function: set_cpu
6        manager: pfsense
7
8  workload_parameters:
9      - name: packetsize
10       values: [64,128,256,512,1024,1500] #
       Bytes
11       function: set_environment_var
12       manager: src
13     - name: flows
14       values: [1, 2, 10, 100, 1000, 10000]
15       function: set_environment_var
16       manager: src
17
18   initialization:
19   - manager: src
20     cmd: 'pkill -9 -f start_traffic_stream.sh'
21   # ...
22
23 primary_workload_parameter:
24     name: packetrate
25     # Values will be chosen in the defined
       interval for this parameter
26     range: [0.1,500] #kpps
27     function: set_environment_var
28     manager: src
29
30   initialization:
31     - manager: src
32       cmd: 'bash start_traffic_stream.sh'
33     # ...
```

**Listing 2** YAML based test descriptor - test configuration space

The sampling heuristic takes the defined parameter ranges into account and will iterate through all combinations in an optimized order. For this reason, the total configuration space is categorized into three sections:

- *resource_parameters*
- *workload_parameters*
- *primary_workload_parameter*

The main reason for this categorization is to guide the sampling heuristic to the metric value to sample next, as will be explained in the coming sections.

Also commands to start or stop the workload are specified in the *Test descriptor*. The commands specified in Line 20 or 32 in Listing 2 are literally executed in a (bash) shell inside the specified VNF. Instead of combining commands in a fixed script, also multiple commands could be specified here as a list. The VNF profiler which is parsing the descriptor, would execute these commands in sequential order as they appear in the descriptor.

In Listing 3, we illustrate the structure of the *Metrics descriptor*. This file is translated to the needed configuration directives for the monitoring framework to gather the required metrics. The list of all the required metrics is given (Line 1). For each given metric, a template should be defined, which maps to the correct metric query (Line 9). Deployment or test specific parameters such as id's should be dynamically filled in the template. The metrics descriptor also defines the probes where the monitoring framework can get the metric values from (Line 21). The Profile service will query all defined metrics from the monitoring database, once a configured workload shows stabilized measurements. (Measurement stability is assessed by the implementation described in [4].) The queried metric values are exported to a file and kept for online analysis by our sampling heuristic. In Subsection 3.3 we will explain the integrated monitoring framework.

It is beneficial if each setting and each exported metric is explicitly mentioned in either the *Test descriptor* or the *Metrics descriptor*. When exporting the test results after the profiling phase, there should be a clear link between the setting/metric name in the exported results and where/how this value was exactly set or measured. Our approach is that the exact name of the setting or metric can be found back in one of the descriptor files. The descriptor files become the reference for the used configuration settings and similarly for what each metric stands for and where/how it is exactly gathered: (i) The test descriptor explains the activated settings regarding the workload generation and resource allocation. (ii) The metrics descriptor explains the metrics gathered from the probes, representing the VNFs performance and current resource usage for example.

```
1  metrics:
2    - sink:cpu
3    - src:cpu
4    - pfsense:cpu
5    - sink:packetrate_receive:eth1
6    - pfsense_packetrate_loss
7    # ...
8
9  definitions:
10   docker:
11     cpu:
12       template: 'sum(rate(
       container_cpu_usage_seconds_total{id="/
       docker/{{ docker_id }}"}[10s]))*100'
13       unit: '%'
14
15     packetrate_receive:
16       template: 'sum(rate(
       container_network_receive_packets_total{id
       ="/docker/{{ docker_id }}",interface="{{
       interface_id }}" }[10s]))'
17       unit: 'pps'
18
19     # ...
20
21 probes:
22   node_exporter:
23     job_name: node_exp_pfsense1
24     scrape_interval: 1s
25     static_configs:
26       - targets:
27           - 'infrastructure.node.url:9100'
28   cadvisor:
29     job_name: cAdvisor_pfsense1
30     scrape_interval: 1s
31     static_configs:
32       - targets:
33           - 'infrastructure.node.url:8080'
34   # ...
```

**Listing 3** YAML based metrics descriptor

The use of the above explained descriptor files, makes it easy to customize and repeat profiling tests. The configuration of monitored metrics, workload and resource parameters is kept very generic to allow a wide applicability in VNF testing.

### 3.3 Cloud-Native Functional Blocks

If we look again at Fig. 2a, we see that several functionalities are implemented by readily available components. The Infrastructure Node should be pre-provisioned. This means that before the profiling tool can operate, the VNFs under test should be pre-deployed on one or more infrastructure nodes. This can be done by a common orchestration framework (e.g. OpenStack, Kubernetes). *Prometheus* is used as monitoring framework and metrics database. Additionally, for every started profiling test, a *Grafana* dashboard is generated, to visually check

the status of the defined metrics being monitored. For each requested metric, the correct Prometheus Query (PromQL) should be given in the Metrics Descriptor, this is a Prometheus specific syntax to retrieve the metric from the database. Prometheus is also configured to send alerts back to profiling service when measurement stability or overload of the traffic src/sink is detected.

To let Prometheus gather the metrics defined in the decriptor, the required probes must be running on each Infrastructure Node:

- *cAdvisor* is a tool to export performance and resource metrics of Docker containers.
- The *Prometheus Node Exporter* does the same for bare metal, or host specific metrics.
- We also use a custom built probe to export VM metrics gathered from KVM and libvirt.

In our setup, the main "ancillary" services are deployed as Docker containers (Prometheus, Grafana, traffic source/sink, probes). The actual VNF under test can be deployed as Docker container or as VM under KVM. Care should be taken that resources (e.g. assigned vCPUs) are well isolated between VNFs under test and other components. Depending on the virtualization method of the VNF (container or Virtual Machine (VM)) we use the configuration options of Docker resp. KVM to isolate the CPU cores between the Device Under Test (DUT) and the traffic sink/source (based on the Linux kernel feature *cgroups*). In the Test Descriptor, separate vCPU cores are being assigned to each VNF. A vCPU smaller than 1 means that a vCPU share smaller than 100% has been allocated. E.g. 0.5 vCPU means that 50% of the vCPU time of one core is allocated to a specific container or VM.

To make sure that the performance of the VNF under test is not bounded by an external factor, we monitor if the sink or source traffic VNF are not overloaded. When this happens, the monitored performance is not bounded by the VNF under test and not representative for its performance profile, and so the performance measurements are invalid in this case. So by detecting the overload in the traffic VNFs, we ensure that tested VNF's performance measurements are not affected.

In the remainder of this article we will present measurement results gathered by the above explained framework and descriptor formats.

## 4 VNFs Under Test

To choose exemplary VNFs for our profiling tests, we looked at some typical use cases defined in [1]. The adoption of 5G technologies enables new possibilities for the telco industry to diversify their network services to new markets. To enhance the security of these services we look at the deployment of a virtual firewall (pfSense). As a large portion of the traffic over 5G will be media based, we also look at the deployment of a virtual streaming server (Nginx). The choice for these two specific VNFs is also to exemplify the generic nature of our presented approach. Different VNFs, which are stressed by different workloads and characterized by different KPIs, are easily testable by our platform and sampling heuristic.

For the tested VNFs in this paper, we consider CPU and network bandwidth as the most important resource metrics, as we experimentally validated these are more likely to become a bottleneck resource than memory. This is also confirmed in [22]. Our measurements also show little to none variation in the memory usage of the VNFs while they are under test. In the next subsection we will discuss each VNF more in detail.

### 4.1 Firewall - pfSense

We use pfSense[1] as a free and open source firewall solution example, deployed as a VM. We stress the firewall by generating multiple unique parallel flows. Also the packetsize is varied. Using the tool *Scapy* we assemble a .pcap file with a stream of packets of varying mac addresses and unique destination IP/port in the packet header. *Tcpreplay* is then used to stream the .pcap file at a given packetrate from the traffic source. There is also an iperf stream running, with an iperf server in the traffic sink. This is used to monitor packet loss. For the firewall to function properly, we need to make sure the ARP table of the VNF contains the mac addresses of the generated packets, so the firewall forwards the packets properly to the traffic sink. This is done by arp spoofing the firewall from the traffic sink. To have an idea of the baseline performance of the firewall, we install no specific firewall rules and let the traffic pass.

*Generated workload metrics*:

- **packetrate**: [0.1-500]kpps. 50 different packetrate values are selectively chosen, spaced evenly along the log scale.
- **packetsize**: [64,128,256,512,1024,1500] bytes
- **flows**: [1,2,10,100,1000,10000] unique parallel flows (with unique IP/port combination in the header).

*Resource metrics*:

- **CPU allocation**: [0.25, 0.5, 0.75, 1, 2, 3, 4, 5] vCPUs

[1] https://www.pfsense.org/

The bandwidth allocation is not a dedicated setting in this test. It is determined by the workload, since we specify the generated packetrate and packetsize up front.

*Performance metric*:
We choose packet loss (%) as the main KPI to reflect the performance of the firewall.

All combinations of above metrics result in 12000 measurement points. If we need about 30sec per measurement to get a stable reading, the total profiling time reaches up to 100h to measure each combination once.

### 4.2 Streaming Server - Nginx

We set up a live streaming service using Nginx[2] , a well known open source, all-in-one load balancer, web server, content cache and API gateway solution. Nginx is deployed in a Docker container. We configure Nginx to accept incoming live movie streams via the Real-Time Messaging Protocol (RTMP [23]) protocol. The incoming RTMP live stream is then transcoded to a specific video bitrate and resolution (Nginx uses ffmpeg for this purpose). Next, Nginx serves the newly encoded movie chunks live, through the HTTP Live Streaming (HLS [24]) protocol. In our test setup, the traffic source sends 1 - 5 movies in realtime to Nginx over RTMP. On the client side, the traffic sink opens many concurrent sessions to Nginx, to download playing live movies over HLS (We use *Locust.io* to emulate the HLS clients and download the stream requests). This use-case exemplifies the situation where a small number of incoming live movies is temporarily cached in an edge server and than streamed with a certain quality to a large number of clients.

*Generated workload metrics*:

- **streams**: [10-5000] parallel client HLS streams. 80 different stream values are selectively chosen, spaced evenly along the log scale.
- **movies**: [1,2,3,4,5] number of different source movie streams, input via RTMP.
- **quality**: [1,2,3,4,5] indicator for the quality of the streams (resolution and video bitrate ranging from 1280x720/2500kbps to 426x240/200kbps).

*Resource metrics*:
The streaming performance is determined by both the available bandwidth and vCPU. It is unpredictable how the balance between cpu time for ffmpeg transcoding and cpu time for serving the movie chunks will be scheduled (as we consider this a black-box VNF). Therefore we have no way to deterministically predict the influence of both the allocated bandwidth and cpu on the KPI. We need to profile the performance with several combinations of allocated vCPU and bandwidth. This also reflects the availability of different flavours to deploy the VNF.

- **flavours**: [(0.5, 0.5), (0.5, 1), (1, 1), (1, 2), (2, 1), (2, 2), (3, 2), (3, 3), (3, 4), (4, 5), (6, 5)] (vCPUs, Gbps). Eleven different flavours to deploy the VNF, defined by their given vCPU and bandwidth allocation and encoded from [0-10].

*Performance metric*:
We choose **lag ratio** (%) as the main KPI to reflect the performance of the streaming server. This indicator is a measure for the risk of "hickups" or lagging during video playback. It is the ratio of downloaded video playback time over the last period. If the video time is less than the waiting time, the playback buffer will empty and the risk of lagging will increase:

$$\text{lag ratio } (\%) = max\left(1 - \frac{T_{video}}{T_{wait}}, 0\right)$$

We measure the lag ratio in a moving average over 20s (we assume 20s buffer time). If the KPI gets above zero, it means that during the last 20s, the playback buffer was addressed because less than 20s of video stream was downloaded. Increasing KPI values mean more buffer time is continuously needed, resulting in video rebuffering and thus "lagging". The HLS protocol will try to keep the lag ratio at zero by varying the size of the served movie chunks and maximizing the bandwidth over all clients.

In order to get a stable measurement, a certain ramp-up tine is need to generate to required number of clients and to let the HLS based streaming stabilize. In our setup this takes up to 100sec per measurement point. To test all above combinations once, takes then over 500h to complete.

We only evaluate the KPIs below 30% packet loss or lag ratio, as we assume that above this threshold the VNF is practically unusable. Therefore there is no need to accurately model the KPI above 30%.

### 4.3 Test Traffic Generation

For our tests, every type of test traffic is started and received via a dedicated script stored inside the traffic source/sink VNF itself. For the Nginx test, all related commands to start ffmpeg to stream the video files, are stored in the script `start_traffic_stream.sh` located in the traffic source VNF. A similar script is stored to start the Locust tool to receive the video stream in the traffic sink VNF. Similarly, for the pfSense

[2] https://www.nginx.com/products/nginx/modules/rtmp-media-streaming/

tests, iperf and TCPreplay commands are also stored in a dedicated script. This script is copied and stored inside traffic source/sink VNF at their build time. In Listing 2 (line 20 and 32) we show how this script can be called and stopped from the descriptor.

### 4.4 Hardware Infrastructure

Our Test Infrastructure Nodes are equal compute nodes with 2x 8core Intel E5-2650v2 (2.6GHz) CPU with Ubuntu 18.04. Linux Bridge is used as the hypervisor switch. We do not change default OS options (e.g. we leave hyperthreading enabled). The Manager Node is a lighter machine: 4 core Intel E3-1220 CPU with Ubuntu 18.04. The main bottleneck resource of the Manager Node is the disk space used by Prometheus, to store all the metrics gathered from various running profiling tests.

The long profiling times of the above introduced VNFs show the need to optimize both: (i) the parallel execution of measurement runs by the profiling framework (as explained in Section 3) and (ii) the sampling strategy to limit the number of needed sampling points. The latter will be explained next.

## 5 VNF Data Analysis

As proposed in [4], we have classified the tested VNF metrics under three groups in the previous section:

- **Workload metrics** reflect the configuration of the incoming traffic to be processed by the VNF.
- **Resource metrics** quantify the allocated resources which determine the cost and processing capabilities of the VNF. For our analysis we express this as *resource usage*, which is the averaged used portion (%) of allocated vCPU and bandwidth.
- **Performance metrics** monitor the Key Performance Indicators (KPIs), to assure that the performance of the VNF remains within the SLA.

From the obtained VNF measurements, we want to derive a model which predicts the performance KPI in function of the given workload and resource allocation. From an abstract and generalized viewpoint, the VNF performance model $f$ can be described as:

$$f(wl, res) = perf \tag{1}$$

where:
$wl$ = input workload (e.g. packetrate, filesize, streams)
$res$ = resource allocation (e.g.number of allocated vCPUs, bandwidth or flavour)
$perf$ = KPI metrics (e.g. packet loss, lag ratio)

Figure 3 shows a subset of our measurements: for each VNF a certain workload configuration is executed on varying resource allocations. The measurements in Fig. 3 confirms these trends (which were also described in [4], but on other VNF examples):

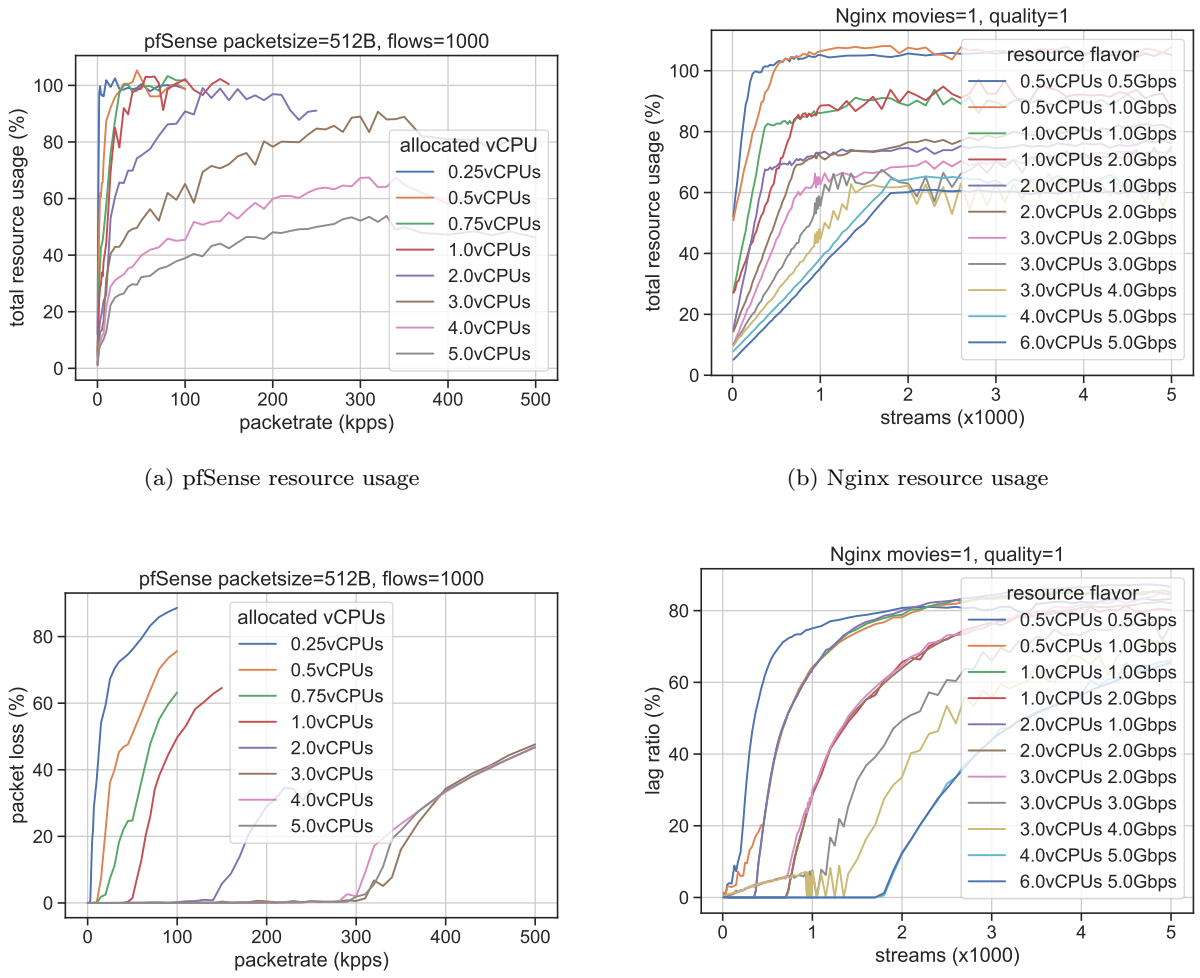
- The resource usage is correlated with the rising workload (on the x-axis) until saturation (Fig. 3a and 3b). Either CPU or bandwidth gets saturated first, which explains why the averaged resource usage can saturate below 100%.
- Before resource saturation, the KPI levels remain stable and flat. When resource contention starts, the KPI levels start to vary more rapidly (Fig. 3c and 3d).

We can also distinguish two other interesting facts from the plots:

- For pfSense (Fig. 3c) we can see that the performance does not increase with more than 3 allocated vCPUs. This points to a deployment limitation where it makes no sense to allocate more vCPUs, because it is not exploited by the VNF implementation.
- When it comes to Nginx (Fig. 3d), we see that some resource flavors have overlapping performance curves. This indicates that the workload is bounded by a common resource limit of those flavors, namely bandwidth in this case.

It is challenging to discover the above mentioned phenomena automatically, without visual inspection of the data plots. A KPI prediction can be made by training the model with the obtained profiled datasets. But a common adagio from the Machine Learning domain is that the model will only be as smart as its training data, meaning that we must provide representative training data in all foreseeable situations. This implies that we must also profile in the regions where resource saturation occurs or where resource flavors overlap, or where the performance is limited by the internal VNF implementation.

In the next subsection we will outline methods to model the performance of the trends shown above. Regarding the accuracy, it is important to note that the breakpoint of the KPI curve is the area of most interest. This is the maximum workload possible by the VNF, just before the performance declines more severely.




(a) pfSense resource usage

(b) Nginx resource usage

(c) pfSense KPI

(d) Nginx KPI

**Fig. 3** Subset of measured VNF metrics under different resource allocations.

### 5.1 Modeling Methods

We look for an appropriate modeling method to predict the KPI values from a given workload and resource configuration, as explained earlier by Eq. 1. A first idea of the trends to be modelled can be seen in Fig. 3c and 3d. From a pure mathematical perspective, we can consider the KPI values to be a response surface, defined by a multi-variate function where the workload and resource allocation metrics are the input parameters. The total input space is multivariate, since all workload and resource metrics can influence the resulting KPI value. We compare several generic methods from the machine learning domain, which are capable of modeling generic, non-linear and multi-variate response surfaces. The used methods have also shown promising applications in regression modeling, where the amount of training samples is limited. We have used the implementations available in the library Scikit-learn [25]. We also include the Interpolation and Curve Fit method, which have shown promising results in [4]. The investigated modeling methods are:

– **Support Vector Regression (SVR)** This method has shown promising results in estimating non-linear relationships using limited, sparse datasets. SVR selects samples to form a flexible tube of minimal radius, symmetrically around the estimated function, such that the absolute values of errors less than a certain threshold ($\epsilon$) are ignored both above and below the estimate. Points outside the tube are penalized and not taken into account for the regression. The hyperparameters of this method are $C$, the penalty parameter, $\epsilon$ and the standard RBF kernel. More details can be found in [26]. The use of SVR for modeling VNF performance has also been applied in [18] with limited

success. We also include it here for verification on our data sets.

- **Random Forest (RF)**: The basic idea behind this method is to combine multiple decision trees in determining the final output rather than relying on an individually built decision tree. Maximum tree depth is set to 10, and the number trees in the forest is 100. The use of decision trees for modeling VNF performance has been investigated in [18], [8] and [19] with promising results. We include the RF method here for verification on our data sets.
- **Gaussian Process (GP)**: This method implements a Bayesian approach to (non-)linear regression. A GP defines a prior over functions, which can be converted into a posterior over functions once it has seen some data. The covariance between training samples is a given kernel function. The kernel function we use: $Constant * RBF + WhiteNoise$. This is a generic kernel function used in many GP examples. One main advantage of using GPs, is that the kernel hyperparameters (RBF lengthscale, noise level, constant) can be learnt automatically via evidence maximisation from the training points themselves, no exhaustive search is needed as with other methods. The key idea is that if the training samples are deemed by the kernel to be similar, then we expect the output of the function around those points to be similar, too. More information on this method is available in [27]. The use of GP for modeling software performance has been proposed in [14]. We also use GP here for verification on our datasets.
- **k-Nearest Neighbors (kNN)**: is a commonly used technique due to its simplicity and often accurate results. In kNN regression, the output value is the average of the values of $k$ nearest neighbors in the input space (weighted by distance). We use the Euclidean distance metric and standardize the input values. For our tests we use $k = 2$.
- **Interpolation method**: Regression is done by interpolating linearly between surrounding samples. The interpolant is constructed by triangulating the input data using Delaunay triangulation, and on each triangle performing linear barycentric interpolation. This method also works in multiple dimensions, so the total workload and resource input space is taken into account to interpolate an intermediate KPI value. This method has been used for VNF modeling in [4] and we also include the method here for verification on our new datasets.
- **Curve Fit method**: This method has been successfully used for VNF modeling in [4] and we reuse the method here for verification on our new datasets. Based on the analysis in [4], we fit the KPI trends to the analytical functions of this form:

$$\begin{cases} f(x)_{\text{non-saturated}} = a + exp\left[b(x-c)\right] \\ f(x)_{\text{saturated}} = 100\left(1 - \frac{d}{x-e}\right), \text{where } x > d+e \end{cases} \tag{2}$$

where the output is clipped in the range [0, 100]:

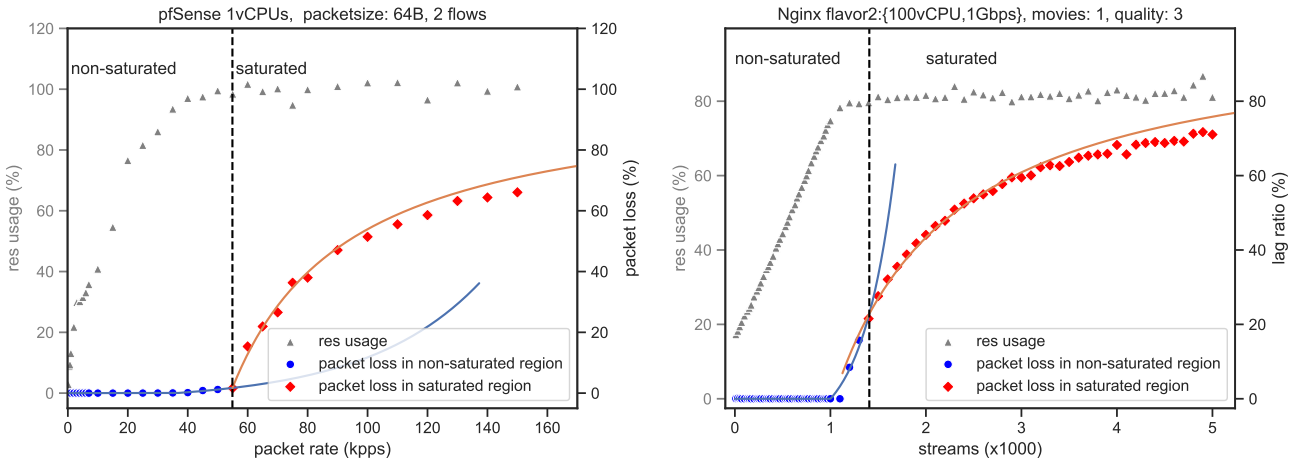
$$\begin{cases} 0 & , where\ f(x) < 0 \\ 100 & , where\ f(x) > 100 \end{cases}$$

The full justification for the functions in Curve Fit method is explained in [4], we summarize here shortly the fitting procedure. Figure 4 shows how the trends in Eq. 2 are fitted to data subsets of both investigated VNFs. The saturated region starts when the workload shows a higher covariation with the KPI than with the resource usage metric. The parameters $a, b, c, d, e$ in Eq. 2 are fitted to the profiled data points in the respective (non) saturated regions. Note that two possible intersections can exist between $f_{\text{non-saturated}}$ and $f_{\text{saturated}}$. When the fitted curve is known, all possible intersections are estimated. For each intersection point, the accuracy (RMSE) of the resulting piecewise model is calculated. Finally, the intersection point which yields the best accuracy is chosen and stored into the Curve Fit model. The clipped output represents that the KPI values (representing packet loss and lag ratio) are limited between [0, 100] %. We further use a weighted curve fitting for the saturated part, to prioritize more accuracy at lower KPI values, just after resource saturation. Since this is the region where most accuracy is wanted.

## 6 VNF Sampling Strategies

In this section, we present our results after investigating different sampling methods. As reference and baseline sampling method, we generate workload samples uniformly (using the values specified in Section 4). The dots in Fig. 4 represent all the samples taken for one specific subset of all generated workloads, described in the figure title. Each of our tested sampling strategies would yield a different set of sampling points, but following the same trend curves. Instead of uniformly spaced samples, our heuristic tries to focus on more interesting parts of the trend curves, like the breakpoint between the (non)saturated regions.

We also check the effect of the sampling heuristic on the accuracy of the modeling methods proposed in previous section. We have executed twice all workload and resource configurations defined in Section 4. We




(a) pfSense example subset

(b) Nginx example subset

**Fig. 4** The Curve fit model exemplified in two example VNF subsets. For each plot, the y-axis on the left is for the saturating resource usage, the one on the right depicts the increasing KPI value. The boundary between (non) saturated regions is where the workload covariance with the KPI becomes larger than with the resource usage.

thus obtain two datasets per VNF, of which one is used as training set and the other as test set. The full factorial test set, containing all possible combinations, counts 14400 samples for pfSense and 22000 samples for Nginx. From the training set, we select data points using the validated sampling heuristics and use these selected samples to train a prediction model for the KPI values. The increasing number of samples taken, is given on the x axis in the plots in throughout this section. We then check the Root Mean Squared Error (RMSE) of the trained model on the test dataset. The reported RMSE in the next sections is then an indication for the accuracy of the model, trained from a limited sample set. We only evaluate the RMSE below 30% packet loss or lag ratio, as we assume that above this threshold the VNF is practically unusable. Therefore there is no need to asses the model accuracy above a KPI threshold of 30%.

### 6.1 Uniform Sampling

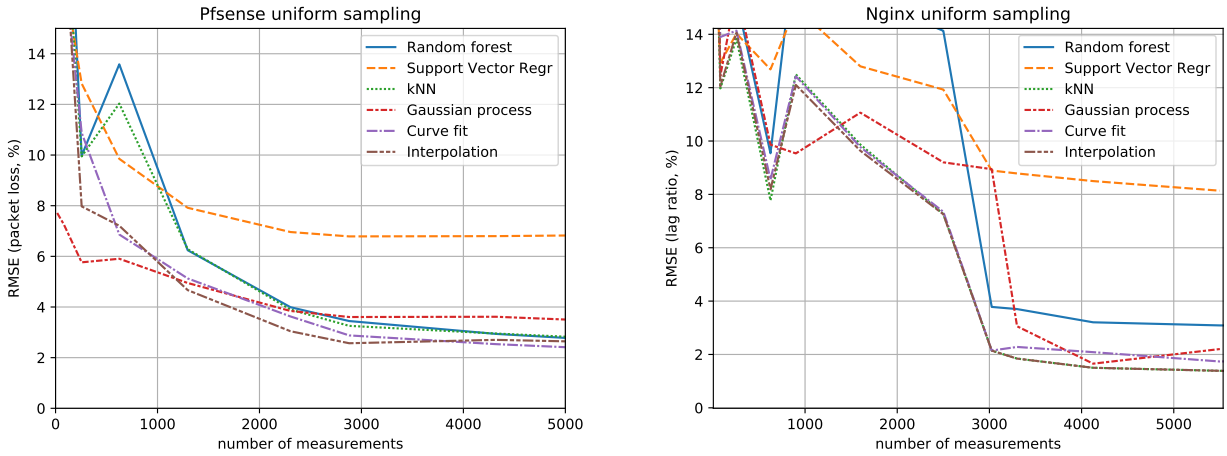
As baseline measurement and benchmark for our sampling heuristic, we first test the accuracy of a generic uniform sampling strategy. For each VNF, we pick resource and workload metric values uniformly in the range described above in Section 4. The result is shown in Fig. 5. On the x-axis, the number of measurements indicates an increasing amount of uniformly chosen values per metric. First we measure one value, then two values for each metric and so forth...

As can be seen in Fig. 5, different modeling methods yield varying accuracy. Support Vector Regression (SVR) proves the least accurate. The tuning of hyperparameters $(C, \epsilon)$ in the SVR model is a tedious task which we do using an exhaustive search and also seems very sensitive to the number of training samples. we experience long training times (tens of minutes with more then 1000 samples). We do not succeed to reach the same accuracy as the other methods. The best performing methods, which we select for further use, are interpolation and curve fit. This is also in line with previous research done in [4].

Uniform sampling is however not an online sampling method, as the number of samples must be chosen in advance. In the next sections, we will try to find heuristics to select samples in an online way, and try to reach the same accuracy with less training samples.

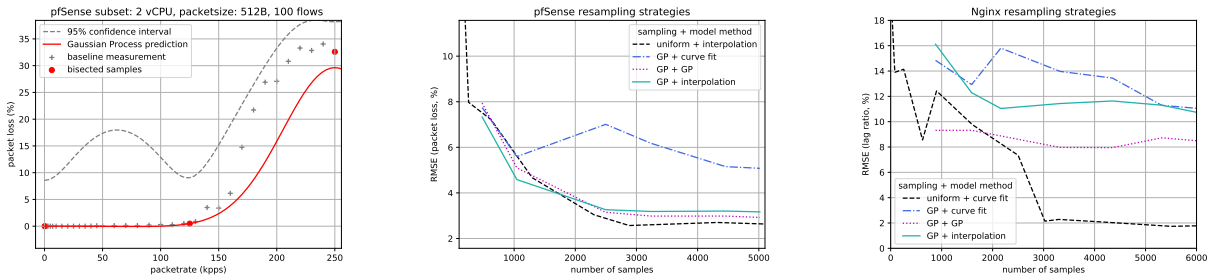
### 6.2 Unsuccessful Strategies

It is worth noting that uniform sampling, exemplified above, is not the least performing method we encountered during our tests. When following the approach outlined in [14], a Gaussian Process (GP) is used to select online which samples to measure next. In fact, GP is a modeling method, but it also produces an estimate of uncertainty in the prediction, a confidence interval for the predicted values as indicated in Fig. 6a. The points where the predicted values have the largest confidence interval, is considered as the most interesting region to sample next. The confidence interval calculated by




(a) pfSense (b) Nginx

**Fig. 5** The accuracy of different modeling methods applied to a uniformly sampled dataset.




(a) Gaussian Process confidence example. (b) pfSense model accuracy using Gaussian Process to select samples. (c) Nginx model accuracy using Gaussian Process to select samples.

**Fig. 6** The Gaussian Process sampling method does not improve the accuracy.

the GP is an indication in which region the uncertainty of the predicted value is larger. For our datasets, this selection method is not optimal, since it favors regions where a flat response is measured, and not the transition zone to the steeper part of the trend.

Fig. 6b and 6c show the effect of a GP chosen sample set on different modeling methods, compared to the uniformly sampled benchmark. We see indeed that GP does not optimally selects samples in our datasets, as the accuracy is worse compared to the benchmark. We can also visualize why some strategies are under-performing. For example, a sampled KPI response surface is illustrated in Fig. 7.

- Sample selection using GP will likely select points in the flat (blue) area of Fig. 7, as explained in previous paragraph.
- On the other hand, gradient-based sampling strategies will likely choose sample points in the steep (green/red) regions of Fig. 7.

For our datasets, the most interesting region to focus on, is where the trend break happens, i.e. where the flat region transforms into the steep region. In the next sections we try to find heuristics which focus on this area.

### 6.3 Feature Selection

In our black-box approach, we try to cover the operational boundaries at the start of the profiling procedure. We measure each combination of the minimal and maximal values of the VNF workload and resource metrics mentioned in Section 4. This comes down to a full factorial sampling space with two levels per factor. This gives us a minimal sample set where an initial analysis can indicate which parameters are important or not. We then use this information to determine which metric configurations to profile next.

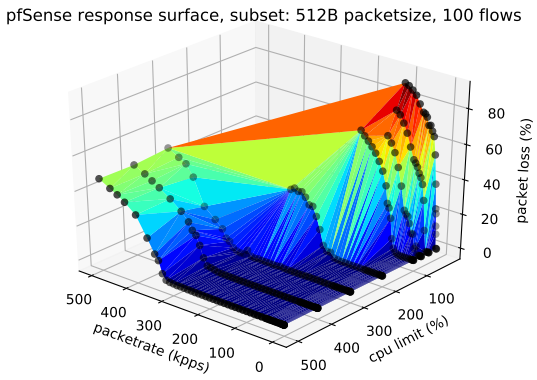
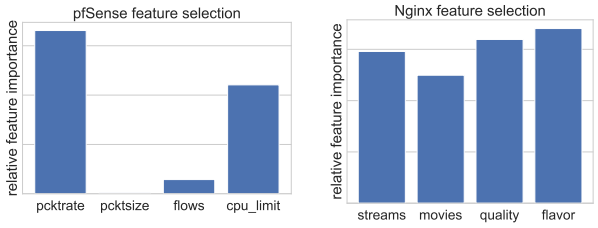



**Fig. 7** A subset of the pfSense dataset to illustrate the response surface of the KPI (packet loss), under different vCPU allocations.

In this first phase of metrics exploration, it is primarily the intention to select the workload and resource metrics ($X$) which have the most influence on the performance metrics ($Y$). Several mathematical methods are generally used to capture the relation between two sets of metrics ($Y = A.X + B$):

- **Multiple Linear Regression (MLR)**: achieves maximum correlation between X and Y, using the well known Ordinary Least Squares (OLS) method.
- **Principle Component Regression (PCR)**: captures the maximum variance in only $X$, using the well known Principle Components Analysis (PCA) method. Then it uses OLS to predict $Y$ from the main components in $X$, calculated by PCA.
- **Partial Least Squares (PLS)**: tries to do both by maximizing the covariance between $X$ and $Y$. The power of this method lies in the fact that influential factors in $X$ can be determined using relatively few samples compared to the other methods. PLS remains robust when the influence on multiple response metrics needs to be assessed (there are multiple columns in $Y$), or when there is multicollinearity between input variables (collinear columns in $X$) [20].

For our purpose, we need a method which can estimate the relation between $X$ and $Y$, from a small initial dataset. Also it is not our main goal in this initial phase to derive a (linear) model. We only need have an idea of the main factors in $X$ that seem to have the most effect on $Y$. We therefore select PLS (also sometimes called Projection to Latent Structures) [20] as most generic method for this use case. The results are shown in Fig. 8, where the height of the bars indicates how much each workload or resource metric influences the KPI variation. The bar heights can be thought of as coefficients of a regression model. But from this small dataset, the only reasonable conclusion for now is that a higher bar indicates a more influential factor for the KPI.




(a) pfSense (b) Nginx

**Fig. 8** Feature selection based on the initial sample set of the profiled VNFs.

- For pfSense in Fig. 8a, we clearly see that workload metrics *packetsize* and *flows* have little to no influence the KPI (packet loss).
- For Nginx in Fig. 8b, we observe that all parameters have a reasonable contribution to the variation of the KPI (lag ratio). This means that we should not be selective in the input metric space here.

We must note that the analysis in the previous subsection takes only the variation between the edge values into account. The sample size is too limited to asses if there are any local extrema in between the edge values. In the next sections we will further investigate where to pick next profiling points.

### 6.4 Primary Workload Metric Selection ($wl_p$)

We further limit the sampling space, by focusing on selected metrics. For each VNF we can prioritize one specific workload metric, which is more likely to vary in real-world traffic. This is the x-axis metric used on the plots in Fig. 3: *packet rate* (pfSense) and *streams* (Nginx). With the VNF deployed in production, we assume that in the most realistic scenarios both resource allocation and workload configuration are relatively stable, while the respective x-axis metrics in Fig. 3 are most likely to vary. We call these the *primary workload metric* $wl_p$. We focus our profiling efforts in such a way, that the VNF model can predict more fine grained at which level of $wl_p$ the performance is outside SLA bounds (using Eq. 1). The other workload and resource allocation metrics are sampled more coarse grained. We favor the respective $wl_p$ metrics during our profiling measurements by picking more samples in their specified range, in order to estimate more accurately the performance breakpoint. For pfSense the $wl_p = packetrate$,

for Nginx: $wl_p = streams$. Intuitively, this corresponds to how we expect a VNF to be generally used: with resource usage mainly driven by increasing packet or request rate.

The coarse grained values, specified for workload and resource allocation metrics apart from $wl_p$, allow the inclusion of "expert knowledge" into the profiling procedure. Next to the edge values, a service developer or tester should include values which are considered interesting to monitor or benchmark. The resource allocation metrics are also likely to be limited by availability. The sampling heuristic will then iterate through the defined workload and resource allocation values in an optimized way.

**Algorithm 1:** Online sampling of workload metric $wl_p$

```
   Data: wl_p boundaries, ε, max
   Result: S_wl_p = sampled wl_p values
1  S_wl_p,old ← the two wl_p boundary values;
2  M_old ← train model with S_wl_p,old;
3  while len(S_wl_p) < max do
4  |  s ← bisect a new wl_p value (Algorithm 2);
5  |  S_wl_p,new = S_wl_p,old.append(s);
6  |  RMSE_old ← RMSE(M_old, S_wl_p,new);
7  |  M_new ← train model with S_wl_p,new;
8  |  RMSE_new ← RMSE(M_new, S_wl_p,new);
9  |  ΔRMSE = |RMSE_old − RMSE_new|;
10 |  if ΔRMSE < ε then
11 |  |  stop while loop;
12 |  end
13 |  S_wl_p,old = S_wl_p,new;
14 |  M_old = M_new;
15 end
16 return S_wl_p,new;
```

Algorithm 1 describes how we select online which values of $wl_p$ to sample next. Also a stop criterion is included, to assess when to stop sampling:

- Line 6: $M_{old}$ is trained *without* the latest added sample. The $RMSE_{old}$ is calculated on how $M_{old}$ predicts all samples, including the new sample.
- Line 8: $M_{new}$ is trained *with* the latest added sample. The $RMSE_{new}$ is calculated on how $M_{new}$ predicts all samples, including the new sample.
- Line 10: $\Delta RMSE$ is a measure of how much the latest sample influences the model accuracy. If the accuracy delta is small enough (threshold $\epsilon$ is used), we assume that the model will not improve further by adding extra samples, and we can stop sampling this particular workload configuration. Throughout our tests we have used $\epsilon = 0.5$.

Note that the modelling method $M$ in Algorithm 1 can be easily replaced. Therefore the algorithm is generic enough to deal with deviating VNF performance trends, (which require modelling methods other than the ones presented in Section 5.1).

In Algorithm 2, we bisect the most interesting workload level to sample next. The goal is to select samples to estimate the point where the KPI trend breaks from flat into a steeper curve, as shown in Fig. 4. Hence, we try to find points around the breakpoint, where the KPI starts to rise.

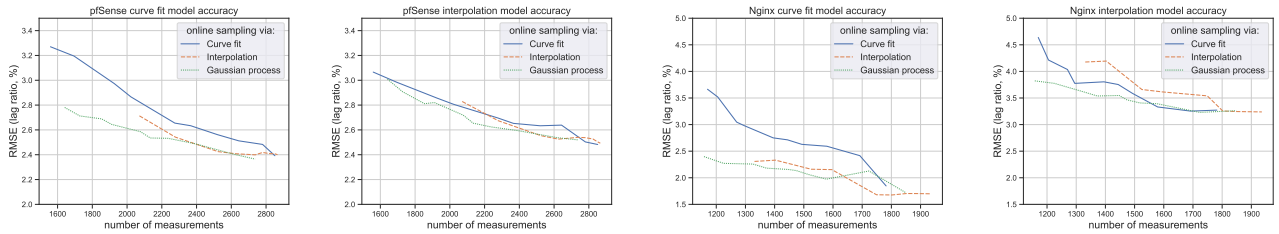
**Algorithm 2:** Bisect next value of workload metric $wl_p$

```
   Data: already sampled wl_p values, KPI_max
   Result: s = newly sampled wl_p value
1  X ← already sampled x values (workload, ascending);
2  Y ← already samples y values (KPI) ;
3  S ← empty set ;
4  foreach sample in (X, Y) do
5  |  Δx = |X[i] − X[i + 1]|;
6  |  Δy = |Y[i] − Y[i + 1]|;
7  |  S.append(i, Δx.Δy)
8  end
9  sort S descending by Δx.Δy ;
10 take i in S with largest Δx.Δy and y[i] < KPI_max ;
11 x_new ← (X[i]+X[i+1])/2 ;
12 y_new ← measure KPI in x_new ;
13 s ← (x_new, y_new) ;
14 return s;
```

- Line 4-8: The already sampled values are grouped per interval. In each interval we calculate $\Delta x.\Delta y$ as a measure how wide and how steep this interval is.
- Line 10: The most interesting interval has both: a large KPI change ($\Delta y$) *and* a wide gap between measured workloads ($\Delta x$).
- Line 11: The next sample is bisected in the selected interval.

In Fig. 9 we investigate which modeling method works best to determine $\Delta RMSE$ (line 2 and 7 in Algorithm 1). For this test we decrease $\epsilon$ from 2 to 0.01. Then for each value of $\epsilon$:

- We filter each total VNF dataset for each unique combination of profiled workload and resource metrics (except $wl_p$). For each of these sample subsets, we use Algorithm 1 to gather samples for $wl_p$.
- In each subset, we compare three methods: curve fit, interpolation and GP, to calculate the $\Delta RMSE$ as stop criterion for the sampling. Each of these three methods thus yields a different number of samples. Note that this is only a one-dimensional model, used




(a) pfSense curve fit model (b) pfSense interpolation model (c) Nginx curve fit model (d) Nginx interpolation model

**Fig. 9** When sampling the workload metric $wl_p$ online, a Gaussian Process (GP) works best to assess when enough samples are taken. The GP produces the best model for the KPI trend, under varying $wl_p$ values, and with the least amount of samples.

here only to calculate $\Delta RMSE$ on training subsets (cf. $wl_p$ on the x-axis and KPI on the y-axis as in Fig. 6a), as $wl_p$ varies while other workload metrics and resource allocation remain fixed.
– When the sampling is completed (threshold $\epsilon$ is reached), we combine the sampled subsets to train both a curve fit (Fig. 9a and 9c) and an interpolation model (Fig. 9b and 9d) on the total input space and check their accuracy.

The GP method emerges here, rather surprising, as the most accurate method in all four tests. Apparently, the GP needs the least amount of samples to capture accurately the KPI trend. Intuitively, this seems to indicate that curve fit and interpolation methods tend to overfit here and require more samples until $\Delta RMSE$ stabilizes. The GP method seems to generalize better in this local one-dimensional model and converges faster. Note that we do not use GP to select the samples here (this would be a bad practice as explained in Section 6.2). Instead we use our own bisection method (Algorithm 2) to select samples and only use GP to calculate the $\Delta RMSE$. Another remark, as shown in earlier measurements, is that the GP method also works less well when the input space is increased with more workload and resource metrics. In this multi-dimensional space, the accuracy of the GP method decreases (we will detail this later in Section 6.6). In the next section we will apply our previous learnings to implement the online sampling heuristic.

### 6.5 Sampling Heuristic Overview

We mentioned earlier that the KPI value can be considered as a response surface of a multi-variate input space defined by the workload and resource metrics. Several generic techniques are available to sample response surfaces, referred to as *Design of Experiments* (DoE). An important learning from this research domain is that edge values are the best starting points to characterize a black-box response surface. Based on this strategy, we have also constructed our sampling heuristic so far:

1. We defined the important workload, resource metrics and their (edge) values.
2. As an initial measurement, each combination of the different edge values is measured (a 2-level, full factorial measurement). A feature selection analysis on this initial sample set reveals which metrics contribute most to the KPI variation and are thus more interesting to sample.
3. One specific workload metric, $wl_p$, is bisected more fine grained, while the other metrics are iterated through fixed pre-defined values.

We want to select in an online way, which next configuration to measure. This means we need to decide from the previous samples, which next workload and resource configuration will reveal thee most information to model the KPI trends. We use following procedure to achieve this: (i) the total sample space is divided into configurations (all possible combinations of workload and resource settings), and (ii) in each configuration, a number of values of $wl_p$ is sampled through bisection. Practically, for our measured VNFs, this means:

– **pfSense**: Each combination of these metrics settings is a configuration: [CPU allocation, flows, packetsize]. In each configuration, we bisect new values for $wl_p = packetrate$, between the given boundary values.
– **Nginx**: Each combination of these metrics is a configuration: [flavor, quality, movies]. In each configuration, we bisect new values for $wl_p = streams$, between the given boundary values.

Per VNF we have thus defined a configuration space. We use Algorithm 3 to select which configuration in this space is most interesting to profile next. In this algorithm, the feature $F_n$ with values $f \in F$, refers to a configuration metric of the VNFs as defined above. We select the next configuration metric value based on previously measured values. The next value is located between the two previously measured values which had

**Algorithm 3:** Online configuration selection

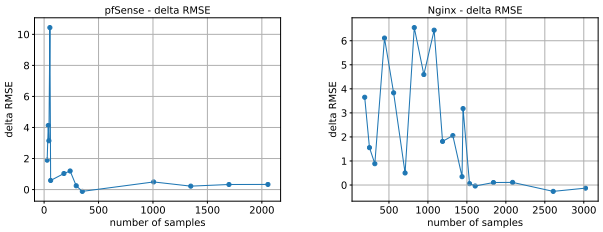
**Data:** $F$ = list of possible values of feature $F_n$
**Result:** $f_{new}$ = new value of $F_n$ to sample

$F_s \leftarrow$ already sampled values of $F_n$ ;
$S \leftarrow$ empty set ;
**foreach** *value* $f$ *in* $F_s$ **do**
  $M \leftarrow$ find all KPI measurements where $f == F_n$ ;
  $m_f \leftarrow mean(M)$;
  $S.append(f, m_f)$ ;
**end**
sort $S$ by $f$ ;
for each interval in $S$ calculate:
  $\Delta m_{f,i} = |m_f[i] - m_f[i+1]|$;
find interval in S where $\Delta m_{f,i}$ is the largest;
$f_{new} \leftarrow$ middle value in $F$ between $f[i]$ and $f[i+1]$ ;
**while** $f_{new} \in F_s$ *or* $f_{new} ==$ *empty* **do**
  take $f_{new}$ in interval with next largest $\Delta m_{f,i}$;
**end**
return $f_{new}$;

the most influence on the mean KPI value (characterized by $\Delta m_{f,i}$). Or in other words, we bisect the new value between the points where the average KPI value showed the highest change. The value returned is then used as the next configuration setting where new values of $wl_p$ are sampled using Algorithm 1.

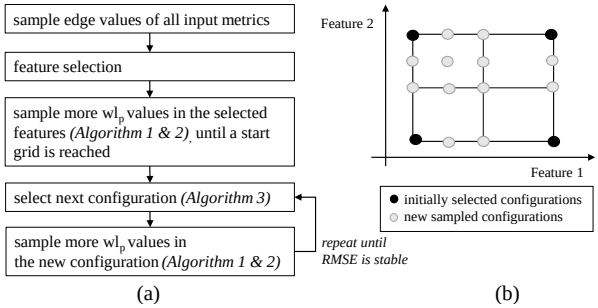
To calculate the RMSE, we have validated our sampling heuristic using an extra test dataset. However, when using the online sampling mechanism on a new VNF, we likely have no test samples available to asses the accuracy of the model being profiled. To alleviate this, we investigate if we can use the $\Delta RMSE$ of the obtained samples so far, as a stop criterion for the profiling procedure, similar to Algorithm 1. In Fig. 10, we calculate the $\Delta RMSE$ when a new workload or resource metric value is added to the profiled measurements. This is an indication if the last obtained samples contributed much to total accuracy of the model so far. We see indeed on the plots that $\Delta RMSE$ stabilizes around the same number of samples where the RMSE stabilizes in Fig. 12. This indicates that $\Delta RMSE$ could indeed be used as a stop criterion, to assess when enough samples are profiled and more samples will not improve the accuracy much more.

Figure 11 summarizes the workflow of our presented sampling heuristic. After feature selection, an initial subset of the configuration space is sampled, as depicted by the black dots in Fig. 11b. The grey dots are added by Algorithm 3. At any point in the sampling procedure, more samples can be added for any of the metrics using the loop in the last two blocks of Fig. 11a. This can be done to increase the accuracy of the prediction model, if there is still budget available for additional profiling time.




(a) pfSense $\Delta RMSE$ (b) Nginx pfSense $\Delta RMSE$

**Fig. 10** Variation in the $\Delta RMSE$, to be used as stop criterion for the sampling.



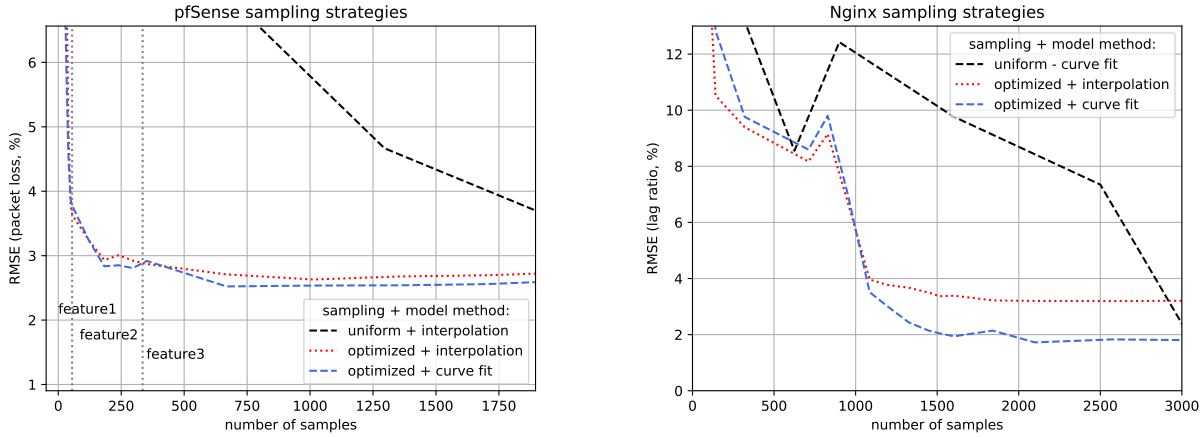

(a) (b)

**Fig. 11** (a) The sampling workflow with (b) simplified representation of the sampled configuration space.

We show the result of the sampling heuristic in Fig. 12. Two selected modeling methods (interpolation and curve fit) are used to indicate the accuracy of the trained model on the test dataset. The heuristic is used to select training samples from the training set. the test dataset is afterwards used to calculate the model accuracy (RMSE). It can be clearly seen that the used sampling heuristic has a positive effect on the needed sample size, compared to the uniform sampling strategy we put forward as benchmark. This beneficial for the total profiling time needed to model the KPI trends of these VNFs. This result also further extends the findings in [4], showing that the curve fit method is also well performing at smaller datasets, with optimally chosen training samples.

In Table 2 and 3 the subsequent rows indicate which metrics have been measured and what the resulting RMSE is (using the best method seen in Fig. 12). Near the bottom of the tables, more workload and resource configurations have been measured, resulting in a lower RMSE and thus better model accuracy.

While sampling the pfSense dataset (Table 2), we can see that most improvement comes after sampling only the first feature (cpu allocation). Sampling the next workload features (flows, packetsize) brings subsequently less improvement. This is aligned with the feature selection done in the beginning of the profiling procedure.




(a) pfSense model accuracy

(b) Nginx model accuracy

**Fig. 12** Optimized sampling strategies compared to uniform sampling. Less samples are needed to achieve a stable and high model accuracy.

| | cpu alloc. (%) | flows | packetsize (B) | packetrate (kpps)($wl_p$) | # samples | RMSE(packet loss) |
|---|---|---|---|---|---|---|
| Edges only | [0.25, 5] | [1, 10000] | [64, 1500] | [0.1, 500] | 16 | (feature selection) |
| Feature1 | **[0.25,...,5]** | 10 | 256 | bisect max 10 values (via algorithm 2) | 56 | 3.77 |
| Feature2 | [0.25,...,5] | **[1,..., 10k]** | 256 | | 352 | 2.92 |
| Feature3 | [0.25,..., 5] | [1,..., 10k] | **[64,...,1500]** | | 2054 | 2.61 |

**Table 2** Configuration picking order for pfSense. Features are sampled in order of significance in Fig. 8. Each feature is completely sampled before the next one.

| | flavor | quality | movies | streams (x1000) ($wl_p$) | # samples | RMSE (lag ratio) |
|---|---|---|---|---|---|---|
| Edges only | [0, 11] | [1, 5] | [1, 5] | [0.01, 5] | 16 | (feature selection) |
| Feature1/2/3 | [0, 11] | [1, 5] | [1, 5] | bisect max 10 values (via algorithm 2) | 40 | 25.0 |
| Feature1/2/3 | [0, **5**, 11] | [1, **3**, 5] | [1, **3**, 5] | | 141 | 12.8 |
| Feature1/2/3 | [0, **2**, 5, 11] | [1, 3, **4**, 5] | [1, 3, **4**, 5] | | 320 | 9.8 |
| Continue to bisect a new value per feature, alternately, until all feature values are sampled... Then also allow more $wl_p$ samples. | | | | | | |
| Feature1/2/3 | [0, **...**,11] | [1, **...**, 5] | [1, **...**, 5] | max 10 values | 1443 | 2.2 |
| Feature1/2/3 | [0, ..., 11] | [1, ..., 5] | [1, ..., 5] | max 50 values | 3025 | 1.8 |

**Table 3** Configuration picking order for Nginx. Features are sampled in order of significance in Fig. 8. New feature values are bisected round-robin over the resource and workload metrics.

As indicated in Fig. 8, cpu allocation is indeed the most important metric in this dataset.

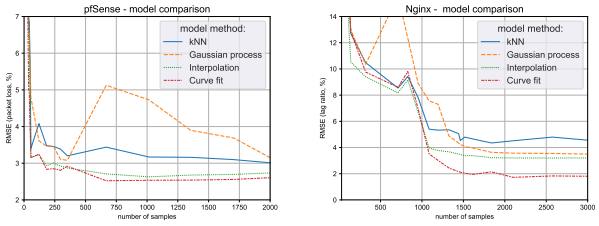
For the Nginx dataset (Table 3), there were no features which jumped out during the feature selection phase. Each metric (flavor, quality and movies) is considered to have significant effect on the KPI. We therefore pick the metrics values round-robin across all input metrics.

It can be noted that also other, more practical, reasons can influence the order in which metrics are selected. For example, it might be more beneficial to limit many resource allocation changes in the sampling procedure. Often, a VNF restart is required to make use of the newly allocated resources such as CPU cores. This reboot (and possible state reconfiguration) might induce a lot of delay in the test procedure. To mitigate this, it can be opted to change the resource allocation as very last feature after all other workload metric values have been sampled.

### 6.6 Modeling Method Impact

In Fig. 13 the same sampled dataset by our heuristic is used to train different modeling methods. It can be clearly seen that the sample set also has an effect on the used model method. During our first tests using uniform sampling, we already found large variations in

accuracy between several modeling methods (as seen in Fig. 5). Using uniform sampling, the interpolation method seemed to provide the best accuracy. Earlier, in Section 6.4, we selected GP as the best method to model a limited one-dimensional sample subset. This does not hold any more when using GP on the total, multi-dimensional input space.




(a) pfSense models (b) Nginx models

**Fig. 13** Comparison of the prediction accuracy of different methods trained with the same sampled dataset.

We selected interpolation and curve fit as most promising methods to be used with our sampling heuristic. In order to use the curve fit method, some expert knowledge or experience is required to define a priori the trends, which need to be fitted to the KPI measurements (as explained in Section 5.1). If such an analytic function cannot be specified, interpolation seems the next best choice.

In Fig. 12 and 13, the curve fit method shows improved accuracy over the interpolation method. The curve fit method has two major advantages:

1. The analytical functions which are fitted have a guaranteed monotonicity. This makes it easier to invert the model and predict a recommended resource allocation from a given KPI and workload target, as no local extrema are present in the modeled KPI. This is explained more elaborately in [4].
2. The curve fit method succeeds to model more efficiently the KPI trend break, happening when resources get saturated at increasing $wl_p$ (see also Fig. 4), using less training samples. It provides better accuracy at low KPI values, where resource saturation is starting but not yet completely dominant.

## 7 Summary and Discussion

In this section, we summarize the contributions of this paper and point out several possible research directions. We have outlined several optimizations to develop and execute a VNF profiling workflow:

- A micro-service oriented architecture to implement a VNF profiling procedure. The benefits of this architecture are that multiple profiling tests can run in parallel, allowing scaled up and faster generation of measurement data. Additionally, a quick, light weight test set up and metric definition is pursued. Also increased robustness can be expected, since the profiling tests run isolated from each other and a failure in one test will not affect other running tests.
- An optimized sampling heuristic to select online which workload and resource allocation to measure next in the profiling procedure. This results in a smaller sample set and thus less profiling time needed to model accurate KPI predictions. Compared to generic uniform sampling, our tests show that up to 5x less samples are needed for the same accuracy.
- We have analyzed the effect of our proposed sampling heuristic on several modeling methods. Careful use is advised for some typical methods used in machine learning such as Gaussian Processes, Random Forest or Support Vector Machines. Other methods can be trained more accurately with the obtained sample sets, as illustrated by our proposed interpolation and curve fit method.

### 7.1 Further Research Directions

The main contribution of our paper is the generic workflow to profile VNFs. This can be further validated by profiling the performance of other types of VNFs also. Similar performance trends on router and firewall VNFs have been found in [4]. But also more non-linear or non-monotonous performance trends can occur [28]. The proposed trend curves in Fig. 4 can then be replaced by other analytic functions or more generic machine learning models. More specifically, customization for other types of VNFs is easily possible by plugging in an appropriate performance trend model $M$ on line 2 and 7 in Algorithm 1. We have focused on the profiling of single VNFs in this paper, but by considering a chained VNF setup as a system on its own, the same profiling principles can be applied. KPIs should then be defined to quantify the performance of the total VNF chain, under a specific set of allocated resources and executed workload.

In our tests, we have profiled a baseline VNF performance, in an environment which is assumed to be representative for production. One possible scenario where this VNF profile can be replicated, is on other identical servers in a datacenter. This means that underlying mechanisms influencing the performance are either included in the profiled model as an invisible factor, or are not profiled at all because they were not at

play during the profiling phase. Investigations regarding collateral effects are not investigated in detail here. It is however possible that a stressing workload on one VNF may impact another one (the so called noisy neighbor effect). For example, the lack of isolation between container allocated resources is well studied topic [29]. Also several benchmarking tests show that both long and short term performance variations exist in shared IaaS environments [30]. To tackle the effects of lacking isolation, VNF profiling should be further extended in combination with several other research domains:

- **Kernel bypassing techniques** offer a greater control over the used CPU resources while forwarding and processing network packets on a generic Linux operating system. Example libraries include DPDK or FD.io. By using these technologies, all packet forwarding tasks are deterministically scheduled to specified vCPU cores. This avoids that the kernel can randomly assign certain packet forwarding tasks to vCPUs already allocated to other VNFs, as explained in [29].
- To adapt pre-profiled VNF models to realtime situations, **online re-training** should be possible during the operation of the VNF. As illustrated with our experiments, pre-profiled data can serve to train a baseline performance model for the VNF. New performance samples, gathered during operation, can then be used to online refine or re-tune the baseline VNF model. The use of online curve fitting during VNF operation is for example shown in [31, 13]. The use of machine learning methods to learn online the performance trends of VNFs is investigated in [32, 33].

The sampling heuristic described in our paper can be adopted to generate reference or baseline training data for multiple VNF related models and use cases. For example, to gather benchmark datasets for VNF anomaly detection or capacity planning.

### 7.2 Conclusion

Our presented profiling framework offers good integration possibilities into cloud-native platforms, this allows to profile VNFs in representative IaaS environments, under realistic workloads. We have selected a set of sampling and modeling methods which we believe to be generic enough, to be applied to a wide range of VNF functionalities. By fitting the measured VNF trends to analytic functions, we add expert knowledge into the VNF performance model. As a result, we see clear improvement of the curve fit method compared to generic parameterized methods typically used in machine learning approaches. This also confirms earlier research results presented in [4].

Our proposed sampling heuristic needs a set of inputs to start from, namely a specified range for workload and resource allocation metrics. One primary workload metric $wl_p$ is bisected in a fine grained way, the other ones will iterate through a list of specified values, which the sampling heuristic will propose online in an optimized way. By using feature selection (using the PLS method) and bisection, the heuristic will select from previous measurements which workload and resource configuration to profile next. This is implemented by our presented Algorithms: 1, 2 and 3. A stop criterium is included to automatically decide when enough samples are gathered to achieve good accuracy. The presented heuristic is well suited to assist in VNF profiling tasks which can be completed faster due to a reduced testing time.

## Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

## References

1. 5G-PPP Software Network Working Group. Cloud-native and verticals' services. Technical report, 5G-PPP Software Network Working Group, 2019. Accessed: 2019-10-01.
2. Angelos Mimidis-Kentis, Jose Soler, Paul Veitch, Adam Broadbent, Marco Mobilio, Oliviero Riganelli, Steven Van Rossem, Wouter Tavernier, and Bessem Sayadi. The next generation platform as a service: Composition and deployment of platforms and services. *Future Internet*, 11(5):119, 2019.
3. Steven Van Rossem, Bessem Sayadi, Laurent Roullet, Angelos Mimidis, Michele Paolino, Paul Veitch, Bela Berde, Ignacio Labrador, Aurora Ramos, Wouter Tavernier, et al. A vision for the next generation platform-as-a-service. In *2018 IEEE 5G World Forum (5GWF)*, pages 14–19. IEEE, 2018.
4. S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester. Profile-based resource allocation for virtualized network functions. *IEEE Transactions on Network and Service Management*, pages 1–1, 2019.
5. Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *ACM/IFIP/USENIX International Conference on*

*Distributed Systems Platforms and Open Distributed Processing*, pages 366–387. Springer, 2008.
6. Vincenzo Sciancalepore, Faqir Zarrar Yousaf, and Xavier Costa-Perez. z-torch: An automated nfv orchestration and monitoring solution. *IEEE Transactions on Network and Service Management*, 15(4):1292–1306, 2018.
7. Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. Nfv-vital: A framework for characterizing the performance of virtual network functions. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 93–99. IEEE, 2015.
8. M. Gokan Khan, S. Bastani, J. Taheri, A. Kassler, and S. Deng. Nfv-inspector: A systematic approach to profile and analyze virtual network functions. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–7, Oct 2018.
9. Manuel Peuster and Holger Karl. Profile your chains, not functions: Automated network service profiling in devops environments. In *Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017.
10. Steven Van Rossem, Wouter Tavernier, Manuel Peuster, Didier Colle, Mario Pickavet, and Piet Demeester. Monitoring and debugging using an sdk for nfv-powered telecom applications. In *IEEE NFV-SDN2016, the IEEE Conference on Network Function Virtualization and Software Defined Networks*, pages 1–5, 2016.
11. Raphael Vicente Rosa, Claudio Bertoldo, and Christian Esteve Rothenberg. Take your vnf to the gym: A testing framework for automated nfv performance benchmarking. *IEEE Communications Magazine*, 55(9):110–117, 2017.
12. Jaehyun Nam, Junsik Seo, and Seungwon Shin. Probius: Automated approach for vnf and service chain analysis in software-defined nfv. In *Proceedings of the Symposium on SDN Research*, page 14. ACM, 2018.
13. Jesus Omana Iglesias, Jordi Arjona Aroca, Volker Hilt, and Diego Lugones. Orca: An orchestration automata for configuring vnfs. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 81–94, 2017.
14. D. Duplyakin, J. Brown, and R. Ricci. Active learning in performance analysis. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 182–191, Sep. 2016.
15. Jayaraman J Thiagarajan, Nikhil Jain, Rushil Anirudh, Alfredo Gimenez, Rahul Sridhar, Aniruddha Marathe, Tao Wang, Murali Emani, Abhinav Bhatele, and Todd Gamblin. Bootstrapping parameter space exploration for fast tuning. In *Proceedings of the 2018 International Conference on Supercomputing*, pages 385–395, 2018.
16. Karel Crombecq, Dirk Gorissen, Dirk Deschrijver, and Tom Dhaene. A novel hybrid sequential design strategy for global surrogate modeling of computer experiments. *SIAM Journal on Scientific Computing*, 33(4):1948–1974, 2011.
17. Ioannis Giannakopoulos, Dimitrios Tsoumakos, Nikolaos Papailiou, and Nectarios Koziris. Panic: modeling application performance over virtualized resources. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 213–218. IEEE, 2015.
18. M. Peuster and H. Karl. Understand your chains and keep your deadlines: Introducing time-constrained profiling for nfv. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 240–246, Nov 2018.
19. Ioannis Giannakopoulos, Dimitrios Tsoumakos, and Nectarios Koziris. A decision tree based approach towards adaptive modeling of big data applications. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 163–172. IEEE, 2017.
20. Kevin Dunn. Process improvement using data, 2019. Accessed 2019-10-01.
21. ETSI. Network functions virtualisation (nfv): Architectural framework. `https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf`, 2014. Accessed: 2020-02-05.
22. Sameer G Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 71–84, 2017.
23. H Parmar and M Thornburgh. Real-time messaging protocol (rtmp) specification. *Adobe specifications, December*, 2012.
24. R Pantos and W May. Http live streaming, rfc 8216. Technical report, 2017. Accessed 2019-10-01.
25. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
26. Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

27. Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006. Accessed 2019-10-01.
28. Ricardo José Pfitscher, Arthur Selle Jacobs, Luciano Zembruzki, Ricardo Luis dos Santos, Eder John Scheid, Muriel Figueredo Franco, Alberto Schaeffer-Filho, and Lisandro Zambenedetti Granville. Guiltiness: A practical approach for quantifying virtual network functions performance. *Computer Networks*, 161:14–31, 2019.
29. Junaid Khalid, Eric Rozner, Wesley Felter, Cong Xu, Karthick Rajamani, Alexandre Ferreira, and Aditya Akella. Iron: Isolating network-based {CPU} in container environments. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 313–328, 2018.
30. Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. Is big data performance reproducible in modern cloud networks? In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 513–527, Santa Clara, CA, February 2020. USENIX Association.
31. Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. vperfguard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 271–282, 2013.
32. Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 19–33, 2019.
33. Johannes Grohmann, Patrick K Nicholson, Jesus Omana Iglesias, Samuel Kounev, and Diego Lugones. Monitorless: Predicting performance degradation in cloud applications with machine learning. In *Proceedings of the 20th International Middleware Conference*, pages 149–162, 2019.

**Steven Van Rossem** received a M. Sc. in Electrical Engineering in 2010 from K.U. Leuven. He started a PhD with the IDLab research group of Ghent University in 2015. His research is situated in the field of Software-Defined Networking and Network Function Virtualization, focusing on scalability and performance profiling of network services. This work contributed to European research projects such as UNIFY, SONATA and NGPaaS.

**Wouter Tavernier** received a M.S. in computer science in 2002, and a Ph.D. degree in computer science engineering in 2012, both from Ghent University. He joined the IDLab, imec research group of Ghent University in 2006 to research future Internet topics. His research focus is on software-defined networking, network function virtualization, and service orchestration in the context of European research projects such as TIGER, ECODE, EULER, UNIFY, SONATA and TANGO.

**Didier Colle** is a full professor at Ghent University. He received a Ph.D. degree in 2002 and a M.Sc. degree in electrotechnical engineering in 1997 from the same university. He is group leader in the imec Software and Applications business unit. He is co-responsible for the research cluster on network modelling, design and evaluation (NetMoDeL). This research cluster deals with fixed Internet architectures and optical networks, Green-ICT, design of network algorithms, and techno-economic studies.

**Mario Pickavet** is professor at Ghent University since 2000 where he is teaching courses on discrete mathematics, broadband networks and network modelling. He is leading the research cluster on Network Design, Modelling and Evaluation, together with Prof. Didier Colle. In this context, he is involved in a large number of European and national research projects, as well as in the Technical Programme Committee of a dozen of international conferences.

**Piet Demeester** is a professor at Ghent University and director of IDLab, imec research group at UGent. IDLab's research activities include distributed intelligence in IoT, machine-learning and datamining, semantic intelligence, cloud and big data infrastructures, fixed and wireless networking, electromagnetics and high-frequency circuit design. Piet Demeester is a Fellow of the IEEE and holder of an advanced ERC grant.