# Automated Monitoring and Detection of Resource-limited NFV-based Services

Steven Van Rossem*, Wouter Tavernier*, Didier Colle*,
Mario Pickavet* and Piet Demeester*
*Ghent University - imec, IDLab.
Email: {steven.vanrossem, wouter.tavernier, didier.colle, mario.pickavet, piet.demeester} @ugent.be

***Abstract*—The growing demand for flexibility and cost reduction in the telecommunication landscape directs the focus of service development heavily to programmability and softwarization. In the domain of Network Function Virtualization (NFV), one of the goals is to replace dedicated hardware devices (such as switches, routers, firewalls) with software-based network functionalities, showing comparable performance when deployed on common servers. In this paper, we discuss how current VNF implementation and deployment strategies impact the efficient monitoring of their resources. In a multi-tenant, NFV-based eco-system, different Service Providers deploy VNFs on a shared infrastructure, where the Infrastructure Provider exposes only VNF specific metrics and little information about the physical hosts where the VNFs are eventually orchestrated. Especially in the situation where datacenters are overcommitted, detecting the risk of e.g. CPU starvation is not straight-forward, when no information from the physical host is available. A new monitoring technique is introduced, based on the skewness of the measured probability distribution of the VNF resource consumption. Our measurements show that this metric is a good indicator for the (un)availability of the required CPU resources in the datacenter.**

***Index Terms*—Monitoring, NFV, Profiling, Overcommit**

## I. Introduction and Motivation

The transformation to a 5G-enabled telecom environment enforces major efforts regarding the softwarization of network functions that were hardware-based in previous generations of telecom networks. The economics behind this transformation are driven by a more flexible and automated deployment of telecom services on common/cheaper hardware, saving both capex and opex related costs. It is changing the way telecom services work, using consumption-based, pay-by-use cost models, enabling businesses to scale resources up and down as demand dictates. In a 5G eco-system, different parties are involved in the service value-chain, each with a different business model. From a Service Provider's view, the goal is to provide sufficient service quality, according to the required Service Level Agreement (SLA), with the least resources possible. From an Infrastructure Provider's view, the requested resources (compute, storage, network) need to be provided as efficient as possible. This means limiting the under-utilization of its servers as much as possible, as decommissioning unneeded server racks can save a lot of money. In this context, it is a common practice for datacenter operators to overcommit their infrastructure, meaning allocated compute instances may get less resources than originally requested. For example, OpenStack has a default CPU allocation ratio of 16:1, meaning that the scheduler allocates up to 16 virtual cores per physical core [1]. Also Openshift (an enterprise version of the Kubernetes container orchestration platform) supports overcommitment for services who need less strict QoS guarantees [2]. This can lead to a situation where different network function processes from multiple tenants are fighting for the same CPU resources, not getting their required share of CPU cycles from the physical host server (a process known as CPU starvation). Similarly, the noisy neighbour effect occurs when an application or virtual machine consumes the majority of available resources and causes performance issues for others on the shared infrastructure. This is illustrated in Fig. I, where an overcommitted datacenter cannot provide all the resources a service has requested. The service's actual resource usage is limited to an unpredictable percentage of the requested total. This causes second-order effects such as packet loss and processing delay, showing overloaded behaviour for the monitored service.

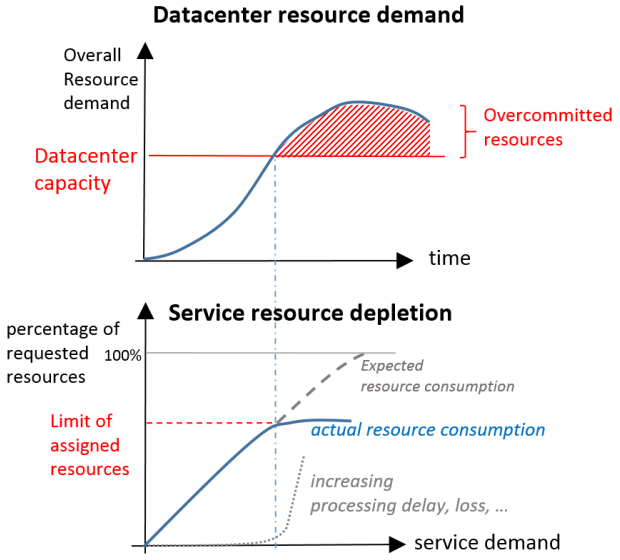


Fig. 1. Overcommitted datacenter, resulting in an unpredictable resource limitation for the service and performance issues.

For security and privacy reasons, the Infrastructure Provider will typically not export any metrics that relate to the shared infrastructure itself, e.g. the total overload of a physical node. Therefore it is not straight-forward for the Service Provider to assess if the deployed service is actually getting

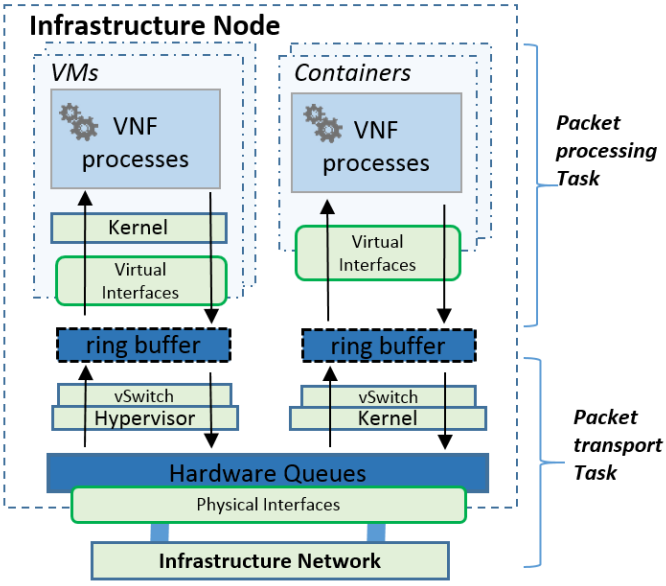


Fig. 2. VNF implementation and deployment on the infrastructure. Resources are allocated to the packet transport and packet processing tasks.

the requested resources. In the next sections, we explain how the implementation and deployment of a network service affects the reservation of hardware resources such as CPU. We introduce a technique that allows an assessment of the service's resource (un)availability, taking into account the Infrastructure Provider's monitoring limitations as explained above.

### A. General VNF implementation and deployment

Advances in the domain of Network Function Virtualization (NFV) yield Virtualized Network Functions (VNFs) that are high-performant, deployable on common servers and scalable to the required performance. The implementation of a VNF, is commonly realized by a virtualization technique (e.g. Virtual Machine or Container). In general, the main function of a VNF process can be summarized as getting packets from the physical network interface, process them and optionally forward them again. This must be done as efficient as possible, to maximize the throughput rate. As indicated in Fig. 2, this implies that packets need to efficiently traverse the hypervisor and/or kernel, before the actual VNF can process them. There are two main tasks contributing to the VNF functionality:

1) **Packet transport**: Transport and buffer the network packets that go in/out the VNF processes. Optimized drivers and libraries such as DPDK, Netmap or pf_ring exist for this purpose. They bypass the kernel and place packets directly into a ring buffer waiting to be processed. Other techniques like SR-IOV allow direct access from the VNF to the hardware queues. In general, this task consumes all resources allocated to the hypervisor, kernel or vSwitch where the VNF is connected to, the NIC hardware and the underlying infrastructure network.
2) **Packet processing**: The VNF process itself, that takes packets of the (ring) buffer, processes them and optionally puts them again in the buffer to be transmitted. This consumes all allocated resources dedicated to the packet processing functionality of the VNF itself.

Resource reservation for VNFs is done according to the two main tasks described above. For the packet transport task, typically one or more CPU cores are exclusively assigned to pushing and pulling network packets from the network device to/from the ring buffer. This CPU reservation ensures that a dedicated and deterministic packet rate is guaranteed for getting packets of the physical interface and making them available to the VNF. For the packet processing task, the CPU resources available to the VNF processes determine how fast packets from the ring buffer can be processed and are the main factor determining the VNF's packet processing rate.

In this paper we focus on the required CPU resources for the packet processing task only. We assume that the packet transport is guaranteed by the Infrastructure Provider. The hypervisor or underlying network is then not the bottleneck, but the CPU resources available to the VNFs are the limiting factor. We further assume that all VNF packet processing tasks are implemented in user-space, allowing distinguishable and monitorable resource usages for all related VNF processes, not 'hidden' as part of the kernel. This resembles an efficient real-world deployment of an NFV-based service. Additionally, if resources in the datacenter are likely to be overcommitted, the VNF performance can only be guaranteed if the overcommitted situation is detected early enough and reacted upon timely (e.g. to limit SLA breaches).

In the remainder of this paper we will elaborate on the statistical techniques that can be used to detect a resource-bounded VNF, by only monitoring the resource use of the VNF itself, not of the hosting physical node. These techniques are then validated by measuring the CPU usage of a VNF on an overcommitted physical node.

## II. Resource Monitoring and Profiling

An important use-case for the correct estimation of resource usage is VNF profiling. This is the procedure where a relation is derived between the service's processing demand and its resource usage, similar to the lower graph in Fig. I. During VNF profiling, several metrics related to performance and resource consumption are measured under load of varying input traffic. When the VNF is under-provisioned, the measured resource consumption is not representative any more and the profile result for this input load needs to be rejected. This also indicates that the performance limit is reached on this physical node and the profiling run can stop.

### A. General Monitoring and Profiling Statistics

For a useful profiling result, it is important that the monitored metrics have a small confidence interval, meaning that the measured value is reproducible within pre-defined tolerances. Existing tools often repeat the measurement a number of times, and then calculate the interval in which the averages

of each test run are located [3] [4]. Following the Central Limit Theorem, the distribution of the sum (or average) of a large number of independent, identically distributed variables will be approximately normal, regardless of the underlying distribution. We can therefore assume that the monitored means of each test run are following a normal distribution, and the sample confidence interval can be easily calculated with known formulas from statistic theory:

$$\left(\bar{X} - t^* \frac{S}{\sqrt{N}}, \bar{X} + t^* \frac{S}{\sqrt{N}}\right) \tag{1}$$

where:

$\bar{X}$ = the sample mean

$S$ = the sample variance

$N$ = the number of samples

$t^*$ = $t_{N-1,1-\alpha/2}$ This is the $(1-\alpha/2)$ quantile of the $t$-distribution for $(N-1)$ degrees of freedom, where $\alpha$ is the confidence level.

Using these formulas, the test run can stop as soon as the calculated confidence interval is reaches a pre-defined boundary (e.g. $95\%$ probability that the metric is located $\pm 2.5\%$ around the sample mean). From Eq. (1) it can be seen that the estimate becomes more precise as the sample size $N$ increases. Since we do not know the variance $S$ of the samples up front, we cannot predict how many samples will be needed for a given confidence interval width. We therefore need to re-evaluate the calculation of the confidence interval as the number of samples increases. To gather *enough* samples for a representative confidence interval can therefore be a time-consuming operation, especially in case of large variances.

In case of a non-normal distribution, Eq. 1 is not valid any more to calculate the (asymetric) confidence interval. In this case, other statistical techniques such as bootstrapping can be used [5]. When the probability is skewed away from normality, calculation of the skewness parameter [6] is a cheaper method to specify the asymmetry of the distribution about its mean. In the next section, we describe how this skewness can be used in an NFV environment to indicate resource starvation.

### *B. Bounded CPU Resource Monitoring*

For a steady ingress traffic flow, we can assume that the VNF processes will consume a steady CPU share, proportional to the traffic rate. This can be derived from Fig. 2, where the VNF processes must be able to process packets from the buffer at the same rate as they are arriving. The assignment of CPU cycles to the VNF process is however not continuous. The operating system's scheduler will actively pre-empt running VNF processes and resume them at a later time. This to allow all processes on the node get a share of CPU cycles over time. For default Linux operating systems this is arranged by the *Completely Fair Scheduler* (CFS). The design goal of this scheduler is to provide each process an equal average CPU share over time. [7]. To characterize the CPU usage of a VNF, we can periodically monitor the incremental CPU time the scheduler has assigned to this VNF:

$$\text{CPU usage} = \frac{\Delta t_{CPU}}{T_{sample}} \tag{2}$$

where:

$\Delta t_{CPU}$ = CPU time counter increase between two samples

$T_{sample}$ = the sample period

Some representative measurements are given in Fig. 3 and Table I. The bin-size is the period during which samples were taken. When observing the different results, we can derive following trends:

- Large bin-size and $T_{sample}$ yield a **quasi-normal distribution** with relatively lower variance, as seen in Fig. 3(a). This can be explained as a larger $T_{sample}$ results in a more averaged value of consecutive CPU time increments by the scheduler. This tends to a normal distribution as stated by the Central Limit Theorem.
- Small bin-size and $T_{sample}$ yield a **right-skewed distribution** in case of sufficiently available resources, as seen in Fig. 3(b). The smaller $T_{sample}$ samples less averaged increments of the CPU time counter, showing more variance. The scheduler has much headroom to add larger CPU time increments if needed, indicated by the right skew of the distribution.
- A **left-skewed distribution** indicates resource starvation, as seen in Fig. 3(c). This can be explained by the scheduler not being able to assign more CPU time beyond a certain limit (also indicated in Fig. I). The CPU time increments show a very large variation bounded by a maximum value. The large skew to the left indicates an increased probability that the scheduler pre-empts the VNF process too soon.

This right-limit of the bounded CPU distribution is difficult to predict, as it is depending on how the scheduler distributes the available CPU time to all processes on the host who share the same CPU (depending on the amount of processes and their priority). The skewness however, seems to be an indication of imminent CPU overcommitment. In order to measure this skewness, a small $T_{sample}$ is needed. Because of the smaller sample time, we can also use a smaller bin-size to gather many samples, speeding up the detection time. The order of skewness seems to be related to the availability of CPU resources. In the next sections we will present some test results to quantify this.

## III. Test Results

We monitor a VNF implemented as a Docker container, which runs a user-space OvS instance (a software switch). By configuring the OvS instance as a user-space switch, we ensure the packet processing tasks are monitorable as indicated in Fig. 2 and Sect. I-A. This serves a our generic model of a VNF whose packet processing capability is CPU-bounded. This container is deployed on a virtual machine with two vCPUs assigned (hence the CPU usage can reach max 2.0 and we assume the host is overloaded when its load average $> 2$). The total CPU time (in nanoseconds) the scheduler has

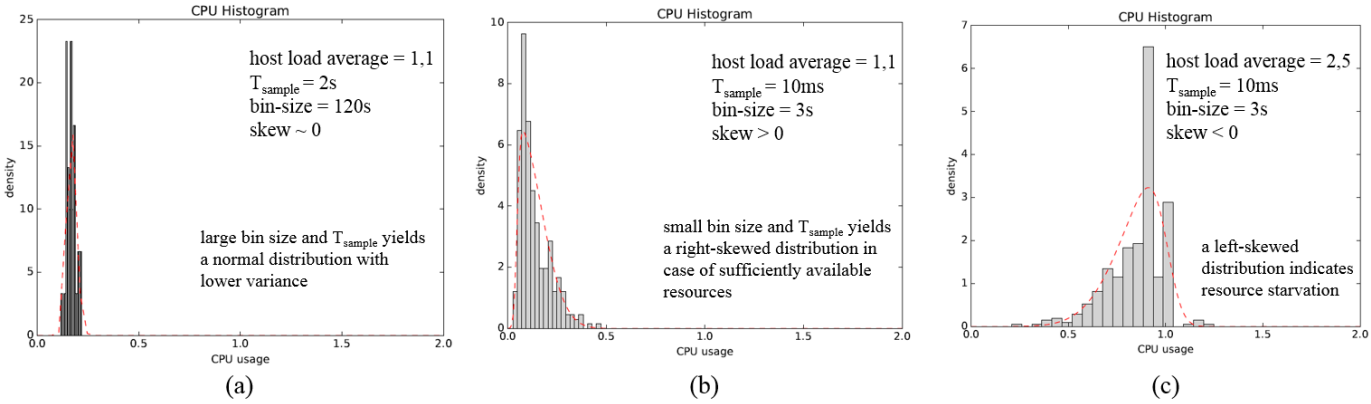


Fig. 3. CPU usage distribution for a VNF under low (b) and high load (c). Left-skewness indicates an overcommitted host.

TABLE I
DIFFERENT CPU USAGE SAMPLING METHODS UNDER LOW AND HIGH LOAD

| host load average (1min) | $N$ | $T_{sample}$ | total sample time (bin-size) | mean ($\mu$) | 95% confidence interval (relative to $\mu$) | skew-ness |
|---|---|---|---|---|---|---|
| 0.78 | 60 | 2s | 120s | 0.233 | [-2.8%, +2.8%] | 0.2 |
| 0.78 | 100 | 10ms | 1s | 0.245 | [-1.8%, +4.6%] | 2.6 |
| 2.6 | 60 | 2s | 120s | 0.858 | [-0.7%, +0.7%] | 0.3 |
| 2.6 | 100 | 10ms | 1s | 0.825 | [-7.6%, +1.0%] | -1.7 |

assigned to this container's processes can be monitored by checking the total CPU time for this cgroup in the pseudo-file located here:

`/sys/fs/cgroup/cpuacct/docker/CONTAINER_ID/cpuacct.usage`

This pseudo-file is available on the host or from inside the VNF (Docker container) itself. It exports the scheduled CPU time counter from kernel to user-space. Alternatively, total CPU time can also be queried using the Docker REST API. The CPU usage is derived using Eq. 2, calculating the timer delta every $T_{sample}$ during a certain period (bin-size). When having access to all the load metrics from the host machine, its overcommitment is more easily detectable. The host load averages (e.g. as reported by *top*) typically show how much the host node was (over)loaded for the last 1/5/15 mins. This is however a slow changing metric, not fit for speedy detection of resource shortages and not generally exported by datacenter operators. Table I indicates that, regardless of the host load average, long-running averages of the CPU usage always yield a quasi-normal distribution. For these monitored values, Eq. 1 can be used to calculate the confidence interval. The downside is that a long time is needed to collect a reasonable quantity of samples. The amount of host overload does play a role when taking short-timed samples of the CPU usage. The faster sampling rate yields more samples which show a skewed probability distribution. Then the asymmetric confidence interval was calculated using the the `scikits.bootstrap` Python library. Figure 4 shows more in detail that the skewness parameter is shifting to a negative value, from the moment the CPU usage stops to increase proportionally with the input load. This shows that $skewness < 0$ can be a trigger to indicate resource unavailability in the infrastructure, before second-order effects such as delay and loss become significant.

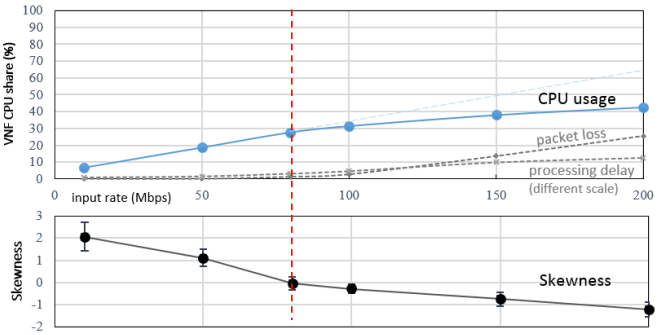


Fig. 4. When the CPU share allocated to the VNF does not increase proportionally with the input load, the host becomes overcommitted. This is indicated by increasing negative skewness.

## IV. INTEGRATION INTO THE MANO FRAMEWORK

In a 5G-enabled environment, NFV-based services are likely to be deployed under the umbrella of a Management and Orchestration (MANO) framework. A VNF will be instantiated with a defined amount of virtual resources. However, NFV allows for the MANO framework to dynamically modify the amount of resources allocated to a VNF, as well as instantiate other VNFs (scaling), as the load requires. In this context, detecting resource starvation is useful for:

- **Overload prediction:** If a resource-bounded VNF is detected, the VNF can be triggered to scale up, or move to another host node, so it can receive more resources.
- **Automated VNF profiling:** As indicated in [8] and [9], the MANO framework can be used to test and quantify the performance of a VNF under bounded resources. During this VNF profiling, multiple performance and resource consumption metrics are measured under varying input load. When the resource limit of the physical host is reached, the measured metrics are not representative any more and the

profile result for this input load needs to be rejected. This also indicates that the performance limit is reached on this node and the profiling run can stop.

The detection mechanism requires that the CPU time allocated to the VNF, or the calculated skewness itself, is exported from the infrastructure. That way, any monitoring entity can assess the resource usage of a VNF.

## V. Related Work

Monitoring the distribution of the traffic rate has been proposed in [10]. There is empirical evidence that for small time-frames, the distribution of the ingress traffic rate is always log-normal. Using the gathered packet rate samples, the parameters of this distribution can be derived. This can give a prediction of the risk that the line rate will be reached for a physical interface. This approach is however less usable for virtual network interfaces, where the maximum rate is in fact CPU-bounded, and thus a variable limit that is depending on the assigned CPU share, as described in Sect. II-B.

An anomaly detection function is proposed in [11]. This approach recognizes a VNF anomaly as a significant statistical deviation from a known set of metrics, gathered during normal VNF operation. Our approach to detect anomalous CPU-usages does not require a pre-defined steady or normal state of the VNF metrics, to compare future samples with. The skewness of the resource usage distribution is an absolute indicator, without the need for a reference sample set.

Methods for pre-deployment testing and performance assessment are described in a report from the ETSI NFV group [8]. Here it is assumed that the VNFs do not compete for the same resources, and have a fixed and guaranteed resource allocation during the test process. This is important: the load generating process (or other VNFs e.g. in a multi-tenant situation) should not compete for the same resources as the VNF under test. This will otherwise impact the performance of the VNF under test, and the performance test results will not be reliable. Detecting any resource competition would be a useful added-value in the context of a real-world deployment, where sufficient resource allocation cannot always be guaranteed.

A VNF profiling controller has been described in [3], [4] to automate VNF profiling. This does not include however a detection mechanism to automatically stop the profiling test run, when the host reaches CPU saturation. A resource starvation detection function would be useful in this context.

## VI. Future work

This paper only investigated the CPU resource usage. Further work will validate also the assumption that other resources such as memory or block I/O show the same usage probability. Also the skewness calculation needs to be optimized, so it can be exported from an infrastructure node with the least overhead possible.

A VNF profiling platform has been described in [3]. An emulation environment, based on Mininet and Docker, allows to deploy and test actual container-based services. The VNFs are orchestrated unto virtual datacenters, which can be overcommitted. This environment was used in this paper and is further developed as part of the H2020 5GPP SONATA project [1]. It will be extended with the resource-limit detection functionality described in this paper, and will be used to test and profile additional VNFs on multiple platforms.

## VII. Conclusion

In a typical NFV-based service deployment, the Infrastructure Provider will not expose any info on the overcommitment of its servers. But especially in VNF implementations, the allocation of insufficient CPU resources can have a major effect on the instant packet processing rate. Our experiments show that the skewness of the CPU usage distribution seems a good *canary* indicator to timely detect this resource overcommitment, before any other second-order effects such as packet loss or processing delay become significant. The proposed detection technique needs only a short sampling time duration (in the order of seconds). By monitoring the proposed skewness metric, the provider will get more confidence on the availability of sufficient resources on the infrastructure nodes where the network service is deployed. This metric can play a role in the context of automated VNF profiling or overload prediction.


## Acknowledgement

This work has been performed in the framework of the SONATA project, funded by the European Commission under Grant number 671517 through the Horizon 2020 and 5G-PPP programs. The authors would like to acknowledge the contributions of their colleagues of the SONATA partner consortium (www.sonata-nfv.eu).


## References


[1] "OpenStack Architecture Guide - Overcommitting Nodes," http://docs.openstack.org/ops-guide/arch-compute-nodes.html#overcommitting, accessed: 2017-01-10.

[2] "OpenShift Configuration Guide - Overcommitting Nodes," https://docs.openshift.org/latest/admin_guide/overcommit.html#configuring-nodes-for-overcommitment, accessed: 2017-01-10.

[3] P. Manuel and K. Holger, "Understand your chains: Towards performance profile-based network service," in *2016 Fourth European Workshop on Software Defined Networks (EWSDN)*.

[4] L. Cao *et al.*, "Nfv-vital: A framework for characterizing the performance of virtual network functions," in *IEEE Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 2015.

[5] F. Wang, "Confidence interval for the mean of non-normal data," *Quality and Reliability Engineering International*, 2001.

[6] D. Joanes and C. Gill, "Comparing measures of sample skewness and kurtosis," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 47, no. 1, pp. 183–189, 1998.

[7] C. Wong *et al.*, "Fairness and interactive performance of o (1) and cfs linux kernel schedulers," in *2008 International Symposium on Information Technology*, vol. 4. IEEE, 2008, pp. 1–8.

[8] "ETSI GS NFV-TST 001 V1.1.1: Network Function Virtualization. Pre-deployment Testing; Report on Validation of NFV Environments and Services," http://www.etsi.org/deliver/etsi_gs/NFV-TST/001_099/001/01.01.01_60/gs_NFV-TST001v010101p.pdf, ETSI Industry Specification Group (ISG) NFV, April 2016, accessed: 2017-01-10.

[9] M. Peuster, H. Karl, and S. Van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE NFV-SDN*.

[10] P. Kreuger *et al.*, "Scalable in-network rate monitoring," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*.

[11] M. A. Kourtis *et al.*, "Statistical-based anomaly detection for nfv services," in *2016 IEEE NFV-SDN*.


[1] https://github.com/sonata-nfv