A Zero-Touch as-a-Service Active Monitoring Framework for Virtualized Network Environments

Alireza Mohammadpour[2], Chiara Lombardo[2], Raffaele Bolla[12], Roberto Bruschi[12], Franco Davoli[12], and Lorenzo Ivaldi[1]

[1]DITEN - University of Genoa, Genoa, Italy,
[2]CNIT – S2N National Lab, Genoa, Italy,

***Abstract*—In order to fulfill the stringent requirements of 5G applications, measuring the performance of the VNFs composing the network slices is crucial to identify potential bottlenecks of the networks. However, since the VNF behavior is time varying and strongly depends on the infrastructure and hosting execution environment, the traditional traffic generators are not suited for the evaluation as their overhead, both in terms of deployment time and code complexity, may affect the results to the point of corruption. In order to overcome this issue, this paper presents a software traffic generator, based on TRex and executed in a VNF, which leverages on an automation framework to provide zero-touch as-a-Service active monitoring. Results show that the impact of this solution on the measured performance is negligible in terms of deployment time as well as required input lines.**

***Index Terms*—Traffic generator, network performance, NFV, 5G**

I. Introduction

The upcoming Fifth-Generation (5G) of radio mobile networks constitutes the second wave of the data revolution in which new innovative services and models can play a key role in the digital transformation towards a new hyper-connected society [1], [2]. 5G-ready applications can be composed of independent, cloud-native *"microservices"* [3] running on individual execution environments (e.g., Virtual Machines – VMs, or containers) deployed across multiple facilities and connected via network slices, which represent logical end-to-end networks, composed of Virtual Network Functions (VNFs), providing specific 5G Network Services (NSs). In order to fulfil the stringent requirements of these applications, the NSs composing a slice must be designed, developed and managed in ways that guarantee a satisfying Quality of Service (QoS). To this end, performance measurement is crucial to identify potential bottlenecks of the networks.

Traditionally, traffic generators have been an essential tool to evaluate the performance of networking environments [4]. Their main job is to inject synthetic packets into the networks so that the characteristic of the generated traffic must be similar to the characteristics of the real one in the networks. Traffic generators are implemented over both hardware and software platforms. Hardware-based solutions are more precise and typically give better performance but they are expensive and in many cases inflexible. Software-based platforms are less precise and suffer from a lack of performance but they are cheaper, more flexible, and often open-source [5].

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

However, monitoring the performance of individual VNFs is a non-trivial task, as their behavior is heavily affected by a number of factors, spanning from the server/datacenter in which they are instantiated, the computational capacity of the execution environment hosting them, as well as their time-varying nature. While several tools are available for the monitoring of network KPIs in the cloud [6], it is hard to find a counterpart for the end-to-end evaluation.

In this paper, we defined a software traffic generator as a service to measure the performance of the virtual networks used in a fully virtual environment. By leveraging on an automation framework to reduce both runtime and setup complexity, our solution exploits the TRex traffic generator [7] in a VNF utilized as a zero-touch as-a-Service for active monitoring the performance of the NSs in different virtual environments.

Tests are performed to evaluate the time of deploying the TRex traffic generator and the ratio of the automation lines compared to the received REST message. Results show that the time ascribable by the automated deployment is negligible enough not to corrupt the measurement and a fully configured traffic generator can be created from a simple REST POST message in a fast, easy and automated way.

This paper is organized as follows. Section II proposes a state of art about different types of traffic generators and categorizes them considering their architectural features and the working layer. Section III introduces a short description of the NFV Convergence Layer (NFVCL) project and describes the TRex blueprint along with the required REST POST message. The evaluation of this study is provided in Section IV, and conclusions are drawn in Section V.

II. Traffic generators

Traffic generators have been a crucial tool to validate network performance for a long time. For this reason, there are plenty of such tools, both proprietary and open source, which have been developed to work at different architectural layers and highlight multiple performance metrics.

So-called replay engines use previous traffic captures as testing streams. Among others, TCPivo [8] is based on commodity hardware and open source software and uses tcpdump

as its main trace collection tool. EAR [9] is also a replay but it is specifically suited for WLAN networks, as it allows transferring a packet-level capture into a sequence of events that follows the IEEE 802.11 protocol. [10] follows a similar approach but its focus is on testing realistic traffic conditions and rate in a cost-effective fashion.

Linux kernel-level generators are commonly used for testing end-to-end network performance. Iperf [11] is a very popular user-level software that allows testing bandwidth, delay jitter and loss ratio characteristics. Brute [12] is specialized for very high bitrates and has extensibility as its main advantage, while Kute [13] is specifically suited for UDP generation and reception.

Synthetic generation, based on statistical models, is also a widespread technique. For example, MGEN [14] generates real-time, packet-level traffic patterns by leveraging on different stochastic models, while D-ITG [15] is a network workload generator that can produce traffic for a wide range of network scenarios by means of a Hidden Markov Model approach.

Finally, it is worth mentioning automatically configurable traffic generators, able to work at a higher level, which exploit live measurements to generate traffic that is very similar to the original one. In this respect, HARPOON [16] is a flow-based traffic generator that can mimic net-flow-based measurements. It can analyze the real measurements and extract the required parameters to create synthetic traffic with characteristics near the original traffic. LiTGen [17] is an open-loop, packet-level traffic generator. It can generate traffics related to the aggregated applications such as web, mail, and P2P by extracting parameters such as session and object characteristics from real measurements.

While the above-mentioned generators, like the majority of the available tools, can cope well with any features of traditional networks, it is very hard to do the same in the presence of NFV. In fact, traditional methodologies cannot isolate the VNF performance from its execution environment (e.g., datacenter/server feature, VM flavors, etc.). Moreover, the time-varying nature of its performance and the impact of orchestration/scaling call for a tool that is highly reliable and can run in an automated fashion.

In this study, we use one of the most powerful traffic generators, TRex, and integrate it with our automation system [1]. TRex is an open-source, low-cost, stateful, and stateless traffic generator fueled by DPDK [7] that generates and analyzes layer 4-7 traffic and provides capabilities of commercial layer 7 tools in one package. It is a traffic generator based on preprocessing and smart replay of real traffic templates which generate the traffic both for the server and client-side at a low cost. It can scale up to 200Gb/s for one network interface card and is able to generate client-side protocols such as ARP, IPv6, IGMP, ICMP, DOT1X, DHCPv6, and some others to simulate a scale of clients and servers. TRex supports two modes, stateful and stateless. While in stateful mode, the basic building block is a flow/application (composed of many packets), i.e., it supports NAT and layer 7 application emulation but, in this mode, it does not support packet field modification and tunneling (in some cases) [7]. In stateless mode it is possible to define a stream with one packet template, to define a program to change any fields in the packet, and to run the stream in one of the following modes: Continuous, Burst, Multi-burst. The stateless mode does not support learning NAT translation, as there is no context of flow/client/server. The stateless mode is much more flexible and enables one to define any type of packet and build a simple program.

III. Automation process

In order to provide automated control of the network and computing resource pools, one needs to properly realize and manage end-to-end network slices and applications. In this way, Telecommunication Operators (TOs) aimed to separate network functions from the purpose-built devices and implement them as software.

Hence, some of the large TOs formed an industry specification group within the European Telecommunications Standards Institute (ETSI) to define the NFV in Oct. 2012 [18]. One of the goals of ETSI is to provide a Zero-touch network and Service Management (ZSM) framework targeting the full automation of end-to-end network services on top of the services provided by ETSI [1]. Nowadays, in a network operator environment, the number of use cases for NFV has increased and the demand for more infrastructure flexibility is a strict requirement. In such a complex network environment, ETSI proposed Open Source MANO, (OSM) which has all the capabilities required to manage this hybrid network infrastructure [19]. The most complex part of MANO is the configuration of NSs.

According to ETSI, MANO includes three main entities, namely: *i*) Virtualized Infrastructure Manager (VIM) that manages and controls the Network Function Virtualization Infrastructure (NFVI) resources such as network, compute, and storage, *ii*) VNF Manager (VNFM) that is responsible for managing multiple VNF instances such as VNF instantiating, updating, searching, extension, and termination, and *iii*) NFV Orchestrator (NFVO) that is mainly responsible for orchestrating NFVI resources and managing the lifecycle of VNFs. In fact, a NS can be composed of several VNFs to be orchestrated and chained by the NFVO. In summary, the main responsibilities of these entities are NFVI management, resource allocation, function virtualization, and so forth [20].

Moreover, the MANO solution should handle all the configuration steps defined as Day-0, Day-1, Day-2, and their vendor-specific differences. The resulting package should include all the requirements, instructions, and elements to achieve these lifecycle stages. Day-0 operations represent the basic instantiation, which requires the configuration of the network interfaces, basic security, and the configuration of a communication framework. Day-1 operations, i.e., service initialization, end when the communication framework is enabled and the NS is accessible on the management network. Day-2 operations correspond to run-time operations, and one of the NFV requirements is that it will still need active management

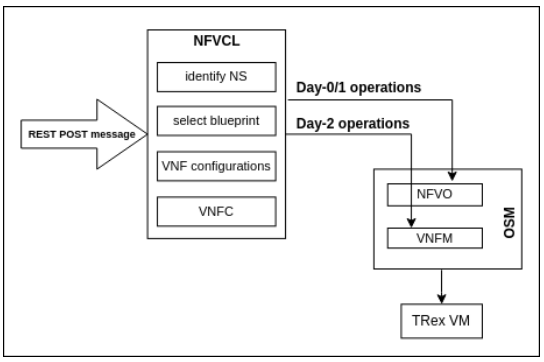

Fig. 1. The process of creating TRex VM through using NFVCL and OSM.

when the NS is active. For instance, one needs to do further configuration as well as continuous health monitoring. [19].

In this work, we utilized a specific micro-service, the NFV Convergence Layer (NFVCL) [1], which provides the level of abstraction required for the flexible and high-level lifecycle management of instantiated NSs, VNFs, and Physical Network Functions (PNFs). The NFVCL applies Day-0/1/2 operations through communication with OSM, and produces and onboards the ETSI SOL006 [21] descriptors of services and related Virtual/Physical/Container Network Functions onto the NFVO, by defining the needed number of virtual links and of virtual resources to be applied (Day-0); then, it asks the NFVO to instantiate the network services and defines the networks and computing resources, as well as the network functions attach points (Day-1), it produces the configuration files and commands for each of the deployed VNFs, and applies them through the VNF Managers (VNFMs) at the NFVO (Day-2).

The foundation of NFVCL lies on the *"network service blueprint"*. As is shown in Fig. 1, a blueprint provides a high-level network template with a pre-determined set of optional/mandatory capabilities that can be customized on the basis on the specific requirements. Inside the blueprints, several software plugins, called VNF Configurators (VNFCs), are made available to provide all information regarding the Day-2 operation primitives for each VNF. Moreover, after the creation the TRex Virtual Machine (VM), the NFVCL uses the OSM VNFM, which can be realised as Juju charms or Helm charts. They are automation tools used for simplifying the deployments of software applications. In our work we developed an Ansible playbook used by the Juju charm to install TRex and its dependencies, create its configuration files, and also configure the TRex VM.

Thus, in this paper, we exploited the concept of the blueprints in NFVCL to develop a ZSM TRex traffic generator. Since the need for measuring the performance of the virtual networks is crucial in designing the VNFs, our traffic generator as a service allows to measure the performance of the virtual devices and virtual networks through only a few lines of REST POST message.

A. The TRex Blueprint

In order to apply the TRex traffic generator in the NFVCL project, one needs to create a specific blueprint. Besides Day-0/1 tasks, Day-2 operations are implemented in an extra module that is called inside the TRex blueprint. As mentioned in Section III, this function uses an Ansible playbook for automation, i.e, it installs and runs a fully configured TRex application inside the generated VM. It also checks the validation of the arriving REST POST message and sets the default values for the parameters which are mandatory for feeding the Ansible playbook but are not defined inside the REST POST message.

It is worth mentioning that TRex uses one management network interface for managing the VM, and at least two extra virtual Network Interface Cards (NICs), belonging to two different networks IP addresses, one for transmitting the generated data and the other one for receiving the transmitted data. The number of the virtual NICs and their names must be provided for NFVCL through the REST POST message. The TRex blueprint not only creates a Linux-based VM but also updates and installs the prerequisites for running TRex. It installs the latest version of the TRex application and creates its directory in the TRex VM and then runs TRex at the first startup. Moreover, TRex blueprint not only saves the output of the TRex first run on the TRex VM root directory, but also returns these results to the terminal.

To be able to run TRex, in particular at the first startup of the VM, TRex needs to be fully configured, and the required information is provided through the REST POST message. This information leads to the creation of two main configuration YAML files named *trex_cfg.yaml* and *cap_customized.yaml*. The former, provided by the TRex blueprint using the Ansible playbook, is responsible for configuring TRex behavior and is located inside the TRex VM. As it is shown in Fig. 2, there is some mandatory information needed to be defined in this file such as *port_limit, version, interfaces, port_info (ip, default_gw)*.

The *port_limit* has to be equal to the number of interfaces

```
- version: 2
  interfaces: ['ens4', 'ens5']
  port_info:
    - ip: 10.0.10.195
      default_gw: 10.0.10.254
    - ip: 10.0.11.35
      default_gw: 10.0.11.254

  platform:
    master_thread_id: 0
    latency_thread_id: 1
    dual_if:
      - socket: 0
        threads: [2,3]
```

Fig. 2. A simple example of generated trex_cfg.yaml file. The generated TRex VM has 8G RAM, 8 vCPUs and two extra NICs which are automatically named as 'ens4' and 'ens5'. The extra NICs IP addresses and their default gateways are taken from input REST POST message.

that are provided by the REST POST message. An even number of extra interfaces should be chosen; otherwise, only the first two interfaces are considered. Regarding these additional networks, the names of the interfaces are automatically generated and are inserted in the *interfaces* section of the *trex_cfg.yaml* configuration file. The created information in *port_info* section of the *trex_cfg.yaml* configuration file is provided by the REST POST message. For more information related to the optional sections, please refer to [7].

One of the optional sections that can be set in the configuration file is *platform*. This section is used to define more information about interfaces pairs and one can provide specific details such as hardware thread id for the "Control thread" and "RX thread" in *master_thread_id* and *latency_thread_id* fields, respectively. Moreover, the "dual_if" section defines information for interface pairs (according to the order in "interfaces" list) in which each section starting with *"- socket"* defines information for different interface pairs including hardware threads.

TRex simulates clients and servers and generates traffic, see Fig. 3, according to the pcap files provided during the installation of TRex VM. Figure 3 shows that TRex contains the client-side and server-side, in which different ranges of IPs for clients and servers must be defined. The traffic generated by the clients and servers should pass through the Device Under Test (DUT), which could also contain a network.

In order to define the type of traffic generated by TRex, one needs to configure a YAML-format traffic configuration file; in our automation process, it is named as *cap_customized.yaml*. The considered range of IP addresses for clients and servers, and the type of traffic, indicated by the pcap files, are defined in this file. The type of traffic can be chosen from *"http", "sfr", "tcp"* and so forth, with respect to the different types supported by the TRex traffic generator. Defining *'sfr'* in the REST POST message means that one needs to generate all types of traffics which can be generated by TRex [7].

All of the parameters mentioned in Fig. 4 can be defined through REST POST messages however, for all of them, a default value is considered to be set since this configuration is mandatory for example, the value of connections per second (cps)=1.1, inter-packet gap (ipg)=1000μs, round trip time (rtt)=100μs, and weight (w)=1. It is worth mentioning that weight indicates the IP generator how to generate the flows. For instance, w=2 means that two flows from the same template will be generated in a burst. The IP address of the ports and their network masks are provided through the REST POST message. These network masks and IPs are also used to configure the VM interface IP address. Moreover, the interfaces' IPs, the clients' IP range, and servers' IP range (the starting and ending IP) are also needed to provide inside the REST POST message. Figure 4 shows an example of *HTTP* customized by the TRex blueprint for the target VM. Each TRex client/server "dual-port" (pair of ports, such as port 0 for the client, port 1 for the server) has its own generator offset, taken from the configuration file. The reason for the "dual_port_mask" is to make static route configuration per

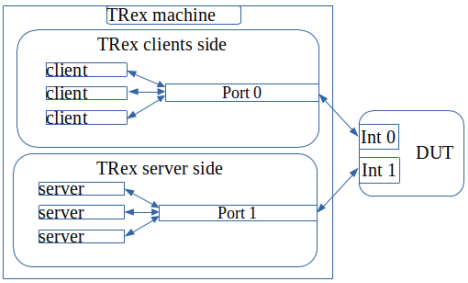

Fig. 3. TRex components.

```
- duration : 0.1
  generator :
        distribution : "seq"
        clients_start : 10.0.10.195
        clients_end : 10.0.10.195
        servers_start : 10.0.11.35
        servers_end : 10.0.11.35
        clients_per_gb : 10
        min_clients : 101
        dual_port_mask : "1.0.0.0"
        tcp_aging     : 0
        udp_aging     : 0

  cap_ipg    : true
  cap_info :
    - name: avl/delay_10_http_browsing_0.pcap
      cps : 1.1
      ipg : 1000
      rtt : 1000
      w   : 1
```

Fig. 4. An example of cap_customized.yaml. The generated TRex VM has 8G RAM, 8 vCPUs and two extra NICs. The type of generating traffic is defined as HTTP.

port possible. With this offset, different ports have different prefixes, see [7] for more information. In our configuration, we leave it as the default value. Moreover, for the rest of the parameters, the default values are selected and one can find a full description of them in [7]. The file is generated from the HTTP sample file in the cap2 folder.

B. REST POST message

Inside the NFVCL project, a REST-API is implemented which accepts all the required configuration fields via a REST message. Figure 5 shows a simple example of the REST POST message. This message has a JSON format and can be divided into three main sections. The first section of the RESPT POST message is dedicated to defining the type of blueprint which in our automation process is TRex. The TRex configuration section, which is needed for running the TRex traffic generator, is already explained in Section III-A. As is shown in Fig. 5, various parameters, mandatory or optional, such as networks' and gateways' IPs, the running duration of the TRex traffic generator, the type of traffic, and the range of clients and servers inside the TRex are provided in this section of the REST POST message. The last section includes a list of data consumed by the VIM in order to create the virtual machine inside the given OpenStack. In particular, the machine will be created inside the defined Open Stack and it

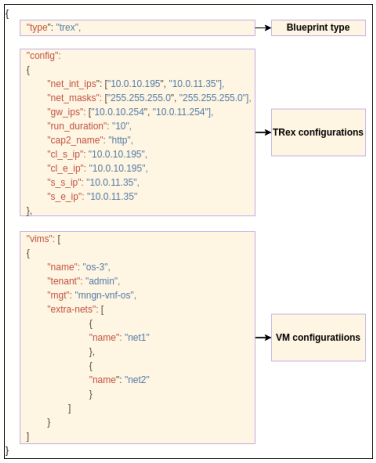

Fig. 5. A REST-API POST message example. It contains three main sections for defining the type of blueprints, TRex configuration information and VM configurations information.

provides the required details, such as the name of the targeted OpenStack and its project name, the management network which is provided for managing the VM; since TRex requires at least two extra network interfaces, these interfaces must be defined also in this section. As noted, the number of interfaces must be even.

IV. Evaluation Results

In order to evaluate our work, we propose two main parameters, firstly the time of deploying the TRex traffic generator and secondly the ratio of the automation lines compared to the received REST message.

A. Time of Deploying

As mentioned before, the goal of this service is to evaluate the performance of the virtual network environment regarding the measurements done by the traffic generator; hence, the starting time of the service is an important parameter. This time highly depends on the hardware and the Internet speed (particularly for Day-2 operations); hence, the required time for the automation process can be divided into different steps.

Firstly, the time required for building packages and onboarding them on OSM, Day-0 operations. Secondly, the time required for configuring resources and NSs and asking OSM to initiate these NSs, Day-1 operations. And finally, the time required for Day-2 operations, i.e., initiating VMs, updating services, installing pre-requisites and TRex, configuring, and running TRex. Figure 6 shows a comparison between the required time for each step when the number of network interfaces changes by 2, 4, 6, 8, and 10 in the REST POST message. All the created TRex VMs have the same configuration, i.e. 8G RAM and 8 vCPUs, and only the number of interfaces has been changed. Figure 7 and Fig. 8 show the spent time for different steps regards the variation of dedicated RAM and vCPUs to the VMs, respectively.

As is expected, at a higher number of interfaces, RAM, and vCPUs, the required time for NFVCL to carry out each step varies slightly. In other words, adding more interfaces, RAM, and vCPUs will not impose considerable extra time to the automation process, while the added automation task in the background could be considerable if one needs to apply the changes manually. It is worth noting that the main time consumption is related to the Day-2 operations which are not dependent on our automaton process in NFVCL and can also vary regarding the TRex running time duration.

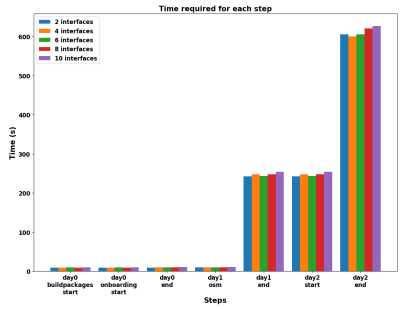

Fig. 6. The required time for each step in the project based on different number of interfaces.

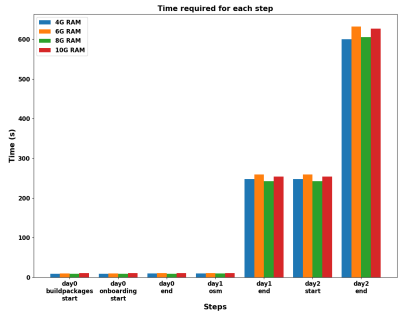

Fig. 7. The required time for each step in the project based on different amount of RAM.

B. Ratio of Automation

This paper focused on the TRex configurations; however other necessary configurations, e.g. NSD and VNFD, are done

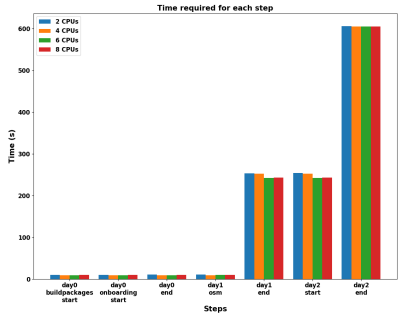

Fig. 8. The required time for each step in the project based on different number of CPUs. There are 2 extra interfaces for each implementation.

automatically through the NFVCL. One may refer to [1] for more information.

In this work, the specification of the VM is assumed to have 8 CPU cores and 8G RAM. Only the number of extra interfaces used for TRex is imported as an input by the REST message and one management network is configured as default. It is worth mentioning that if the number of extra networks increases, the dedicated virtual CPUs also increase automatically. However, in this work, we assumed to have only two extra network interfaces used for TRex VM besides the management network.

Since the implementation time highly depends on the hardware of the server, we propose another measurement value named as the ratio of automation, that is, the ratio between the number of input lines in the REST POST message and the automated configured lines for creating NSD, VNFD, trex_cfg.yaml, and customized_cap.yaml files.

Table I shows the number of configured lines for each configurations files separately. The number of lines in the REST POST message that are mandatory for running TRex can be defined almost in 15 lines, as shown in Fig. 5. Therefore, the ratio of automation can be approximately 56.5, meaning that for each line of the REST POST message we will have 56.5 lines of automaton. Note that in our computation the lines required to install TRex and its prerequisites are omitted from this ratio, i.e., only the created lines for configuration files are considered. Regarding the ratio of automation, one can find that by providing a simple REST POST message, a fully configured traffic generator will be created and the results of generated traffic will be received. These results can be used to analyze the DUTs, which is a concern in the networking design and debugging.

TABLE I
NUMBER OF CONFIGURED LINES IN EACH CONFIGURATIONS FILES.

Input lines	Output lines				
POST message	NSD	VNFD	trex_cfg	customized_cap	ansible playbook
15	527	97	14	20	189

V. CONCLUSION

This paper proposed the TRex traffic generator as a VNF implemented through an automation process via NFVCL. It provides both the ability to configure TRex through REST POST messages and return its output and access to the created VM via SSH or removing the VM from OS. Since the required time to run is low, and also the difficulty of setting up this VNF (shown by a high ratio of the automation) is very low, this VNF can be utilized as a zero-touch as-a-Service active monitoring in different virtual environments to measure the performance of the NSs.

ACKNOWLEDGEMENT

This work has been supported by the Horizon 2020 5G-PPP Innovation Action 5G-INDUCE (Grant Agreement no. 101016941)

REFERENCES

[1] R. Bruschi, J. F. Pajo, F. Davoli, and C. Lombardo, "Managing 5g network slicing and edge computing with the matilda telecom layer platform," *Computer Networks*, vol. 194, 2021.

[2] D. Soldani and A. Manzalini, "Horizon 2020 and beyond: On the 5g operating system for a true digital society," *IEEE Vehicular Technology Magazine*, vol. 10, no. 1, pp. 32–42, 2015.

[3] D. Szabó, F. Németh, B. Sonkoly, A. Gulyás, and F. H. Fitzek, "Towards the 5g revolution: A software defined network architecture exploiting network coding as a service," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 105–106, 2015.

[4] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, pp. 275–287, 2015.

[5] A. Botta, A. Dainotti, and A. Pescapé, "Do you trust your software-based traffic generator?," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 158–165, 2010.

[6] "Prometheus - from metrics to insight - power your metrics and alerting with the leading open-source monitoring solution." https://prometheus.io.

[7] "Trex realistic traffic generator." https://trex-tgn.cisco.com/.

[8] W.-c. Feng, A. Goel, A. Bezzaz, W.-c. Feng, and J. Walpole, "Tcpivo: A high-performance packet replay engine," in *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pp. 57–64, 2003.

[9] C.-Y. Ku, Y.-D. Lin, Y.-C. Lai, P.-H. Li, and K. C.-J. Lin, "Real traffic replay over wlan with environment emulation," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2406–2411, 2012.

[10] T. Ye, D. Veitch, G. Iannaccone, and S. Bhattacharya, "Divide and conquer: Pc-based packet trace replay at oc-48 speeds," in *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pp. 262–271, IEEE, 2005.

[11] C.-H. Hsu and U. Kremer, "Iperf: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*, Citeseer, 1998.

[12] N. Bonelli, S. Giordano, G. Procissi, and S. Raffaello, "Brute: A high performance and extensibile traffic generator," in *Int'l Symposium on Performance of Telecommunication Systems (SPECTS'05)*, vol. 1, pp. 222–227, 2005.

[13] Zander, Sebastian and Kennedy, David and Armitage, Grenville, "Kute A high performance kernel-based udp traffic engine," in *Swinburne University of Technology. Centre for Advanced Internet Architectures*, 2005.

[14] "Multi-generator (mgen) network test tool." https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/.

[15] S. Avallone, S. Guadagno, D. Emma, A. Pescapè, and G. Ventre, "D-itg distributed internet traffic generator," in *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pp. 316–317, IEEE, 2004.

[16] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 68–81, 2004.

[17] C. Rolland, J. Ridoux, B. Baynat, and V. Borrel, "Using litgen, a realistic ip traffic model, to evaluate the impact of burstiness on performance," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pp. 1–8, Citeseer, 2008.

[18] "ETSI, NFV white paper1 ." Available at https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[19] "Open Source MANO." Available at https://osm.etsi.org/wikipub/index.php/Main_Page.

[20] B. Yi, X. Wang, K. Li, M. Huang, *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[21] "ETSI GS NFV-SOL 006, 'Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on YANG Specification', V2.7.1, 2019." https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/, 2019.