17

1 Introduction

1.1 Motivation

Emerging technologies such as SDN and NFV can be considered the great promises that would drastically change the development and operation of computer networks. However, its enabling technologies such as virtualization still pose challenges as for performance, reliability, and security [Han et al., 2015]. Closed hardware solutions are easier to pass on Service Layer Agreement, since they have a much more predictable behavior. Since it is expected that virtualization will affect performance negatively, these VNFs have to keep their degradation as small as possible, therefore, guaranteeing the Service Layer Agreements on emerging scenarios then becomes a harder question. There is a demand for more reliable methods to ensure SLAs over different types of loads.

It is already a well-known fact that the type of traffic used on performing tests matters, studies show that a realistic Ethernet traffic provides different and more variable load characteristics on routers [Sommers & Barford, 2004], even with the same average bandwidth consumption. This fact indicates that tests that employ constant bit rate traffic generator tools are not enough for a complete validation of new technologies. There are many reasons for this behavior, which includes burstiness and packet sizes.

A burstier traffic can cause packet more buffer overflows on the network [Cai et al., 2009] [Field et al., 2004] [Kushida & Shibata, 2002], thus degenerating network performance[[1]](#footnote-2)1 and measurement accuracy [Bartlett & Mirkovic, 2015] [Vishwanath & Vahdat, 2008]. Another key question is how applications will deal with packets, since it is a well-known fact that applications have a huge performance degradation in processing small packets [Srivastava et al., 2014]. A realistic synthetic traffic must not have a single packet-size but rather use a distribution [Castro et al., 2010].

Furthermore,

Rrealistic workload generators are an essential security research [Botta et al., 2012], since generating realistic workloads is an important measure to the evaluation of firewall middleboxes. It includes studies of intrusion, anomaly detection, and malicious workloads [Botta et al., 2012]. Since on traditional hardware-based types of middleboxes the impact of realistic traffic is not negligible, we can expect this impact over virtualized middle-boxes to be even larger, due to the extra overhead of a virtualization layer.

Another critical point is the flow-oriented operation of SDN networks, in which each new flow arriving on an SDN switch demands an extra communication load between it and the controller, which may create a bottleneck between the switch and the controller. Also, the SDN switches have a flow-oriented operation and, since its operation relies on queries on flow tables, a stress load must have the same flow properties of an actual Internet Service Provider. Because of this, there is a demand for studying the impact of realistic traffic on this new sort of environment, as how VNFs and virtualized middle-boxes and SDN testbeds will behave if stressed with a realistic traffic load in comparison to a constant rate traffic is a relevant subject.

Fuatures



1.2 Related Work

The open-source community offers a huge variety of workload generators and bench-marking tools [Botta et al., 2012] [Molnár et al., 2013] [Srivastava et al., 2014] [Kolahi et al., 2011]. Most of these tools were built for specific purposes and goals, so that each uses different methods of traffic generation and enable control of different features, such as: throughput (bit-s/bytes per second, packets per seconds); packet-sizes; protocols and header customization; pay-load customization; inter-packet times; On/Off periods; starting and sending time; and emulation of applications, such as Web server/client communication, VoIP, HTTP, FTP, p2p applications, and many more.

Some traffic generator tools offer support emulation of single application work-loads, however, that does not correspond to real complex scenarios, whereas other tools work as packet replay engines, such as TCPreplay and TCPivo. While it is possible to produce a realistic workload at high rates by that way, it comes with some issues. Firstly, the storage space required becomes huge for long-term and high-speed traffic capture traces, lastly, obtaining good traffic traces is sometimes hard, due to privacy issues and fewer good sources.

Many tools support a larger set of protocols and high-performance, such as Seagull and Ostinato. Others are also able to control inter-packet times and packet sizes using stochastic models, like D-ITG [Botta et al., 2012], sourcesOnOff [Varet, 2014], and MoonGen. All of them offer a complex configuration framework, but their customization is all up to the user, so there is no simple way for the user to create a synthetic realistic traffic scenario.

We also have a variety of APIs available which enable the creation of traffic and custom packets, all of which include low-level APIs, such as Linux Socket API, Libpcap, Libtins, DPDK, Pcapplusplus, libcrafter, impacket, scapy, and many others. These APIs provide a finer control and customization of each packet, and can be used to implement traffic generators. For example, Ostinato and TCPreplay uses Libpcap, and MoonGen uses DPDK. In addition, many of the listed traffic generators provides their own API, such as Ostinato Python API, D-ITG C API, and MoonGen LUA API.

We have a



There is a large variety of open-source tools available for custom traffic generation, but reproducing a realistic traffic scenario is a hard matter. Selecting the right framework, a good traffic model, and the right configuration is by itself a complex research project [Bartlett & Mirkovic, 2015]

e [Leland et al., 1994]. Since it usually is not the main goal of the project, but just a mean, many times, a simplistic and unrealistic solution is selected due to the limited resources such as time and labor. Reproducing a realistic traffic through these tools is a manual process and demands an implementation of scripts or programs leveraging human (and scarce) expertise on network traffic characteristics and experimental evaluation.

There are few solutions on the open source community available aiming to fill this gap. Tools like Swing and Harpoon, for instance, uses capture traces to set internal parameters, providing an easier configuration. Moreover, Swing uses complex multi-levels models that are able to provide a high degree of realism [Vishwanath & Vahdat, 2009], however, they have their issues as well. Harpoon does not configure parameters at packet level [Sommers et al., 2004] and is not supported by newer Linux kernels, which may be a huge problem with setup and configuration. Swing [Vishwanath & Vahdat, 2009] aims to generate realistic traffic, although focusing on background traffic, and high throughputs is not an objective of the application [Vishwanath & Vahdat, 2009] [Bartlett & Mirkovic, 2015]. Because its traffic generation engine is coupled to its modeling framework, you cannot opt to use a newer/faster packet generation library. The only way of replacing the traffic engine is by changing and recompiling the original code, which is clearly a hard task [Bartlett & Mirkovic, 2015], besides being an action prone to error. Again, we fall under the same issue, a complex task that is usually not the goal of the project.

Another matter is the large variety of tools, along with different methods of configuration and limitations. To create a custom traffic, a user must read large manuals and custom-design scripts. One of our bigger proposals is to create a tool able to automatically do these processes, so the user may design his custom traffic by creating his own Compact Trace Descriptor, and create a traffic using many different tools, like Ostinato, D-ITG, Iperf, whilst not requiring to worry about how to proper configure each of them. In Table 1, we summarize our current scenario.

Table 1 – Current scenario of traffic generation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sollution | User Space | Autoconfigurable | Realistic Traffic | Traffic Custumization |  | Extensibility |
| Harpoon | no | yes | yes | yes |  | no |
| D-ITG | yes | no | yes | yes |  | no |
| Swing | yes | yes | yes | no |  | no |
| Ostinato | yes | no | no | yes |  | yes |
| LegoTG | yes | no | no | yes |  | yes |
| sourcesOnOff | yes | no | yes | yes |  | no |
| Iperf | yes | no | no | yes |  | no |

1.3 Problem Statement

Based on what we have stated, we are going to formalize our research targets and define the request list to fulfill these gaps. Our main research targets in this work are:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

* Survey open-source Ethernet workload tools, addressing different features of each one. In this step, we want to evaluate the existing ready to use solutions for network researchers and developers, in addition to what can be used and integrated into our project as part of our solution;
* Define what a realistic Ethernet traffic is and provide a set of metrics to measure the realism and the similarity between original and synthetic traffic;
* Research the main approaches and methods found on the literature to create a model Ethernet traffic, and for realistic traffic generation;
* Create a general method for modeling and parameterizing Ethernet traffic, aiming to mimic any traffic provided as input;
* Create a software able to learn metrics from real network traffic, based on the that reproduce its workload with similar (but not the same) characteristics, avoiding the storage of pcap files.

Next, we will define a requirement list for the tool we are going to develop and present in this research:

* Auto configurable: It must be able to extract data from real traffic and store it in a database, and then use it to parametrize its traffic model. It must be able to obtain data from real-time traffics and from pcap files;
* Technology independent: It must have a flow-based abstract model for traffic generation, not attached to any specific technology;
* Human readable: It must produce a human-readable file as output that describes our traffic using our abstract model. We call this kind of file a Compact Trace Descriptor;
* Extensibility: The traffic modeling and generation must be decoupled. Ideally, it must able to use as traffic generator engine any library or traffic generator tool;
* Simple usage: It must be easy to use. It should take as input a Compact Trace Descriptor, just as a traffic replay engine (such as TCPreplay) would take a pcap file;
* Traffic generation programmability: It must have what we call “traffic generation programmability”. The compact trace descriptor must be simple and easy to read. That way, the user may want to create his own custom traffic, doing it in a platform agnostic way. The traffic programmability must be flow-oriented;
* Flow-oriented: The traffic modeling and generation must be flow-oriented. Each flow must be modeled and generated separately.



1.4 Document Overview

In this introductory chapter, we presented an abstract state of the art, a problem statement, and then proposed an approach for research and requirements for development.

In Chapter Section 2, we go more in-depth on some subjects mentioned here. First, we present an extensive survey on open-source traffic generator tools, and then we summarize the benefits and features supported by each one. After, we offer a brief review of essential topics on realistic traffic generation. We also define some central concepts we are going to use in this work, such as self-similarity and heavy-tailed functions. Then, we discuss some techniques on validation of traffic generator tools and some practical examples. We also analyze some related works.

Chapter 3 introduces the methodology used in our project. We describe SIMITAR low-level requirements and define an architecture along with its algorithms. We also present its classes design and explain how SIMITAR can expand to any traffic generator engine or library with an API, CLI interface. We use the Iperf as an example as we explain its operation and suggest some utilization cases.

In Chapter 4, we go deep in some subjects pointed at the previous chapter. We present how the modeling process works using a defined data set (which we are going to use in the rest of the work). We also show some evaluation methods to check the modeling quality. Besides that, we also describe our used and developed algorithms.

In Chapter 5, we define a set of metrics based on previous tests on validation of traffic generators found in the literature. Here, we focus on packet, flow, and scaling metrics, in addition to testing SIMITAR in an emulated SDN testbed with Mininet, using OpenDayLight as controller [The OpenDayLight Platform, 2017].

Finally, on chapter Conclusion and Future Work 6, we summarize our work and highlight future actions on how to improve SIMITAR on realism and performance.

1. * Features such as packet-trains periods and inter-packet times affect traffic burstiness

   [↑](#footnote-ref-2)