Journal of Internet Technology and Secured Transactions (JITST), Volume 8, Issue 1, 2020

A Comparative Study of Traffic Generators: Applicability for Malware Detection Testbeds

Matthew Swann, Joseph Rose, Gueltoum Bendiab, Stavros Shiaeles, Nick *Savage*
Cyber Security Research Group, University of Portsmouth, UK

Abstract

Network traffic generators are invaluable tools that allow for applied experimentation to evaluate the performance of networks, infrastructure, and security controls, by modelling and simulating the communication packets and payloads that would be produced by machines and devices on the network. Specifically for security applications, these tools can be used to consistently simulate malicious activity on the network and test the components designed to detect and mitigate malicious activities, in a highly reliable and customisable way. However, despite the promising features, most of these tools have some problems that can undermine the correctness of experiments. The accuracy of the simulation results depends strongly on the performance and reliability of the used generator. Thus, in this paper, we investigate the performance and accuracy of three of the most reviewed network traffic generators in literature, namely Cisco TRex, Ostinato and Genesids. Mainly, the comparative experiments examine the strengths and limitations of these tools, for malicious traffic- which can help the research community to choose the most suitable one to assess the performance of their networks and security controls.

1. Introduction

Network Traffic Generators (NTGs) are vital tools in the networking and security fields [1, 2, 3]. They mainly focus on creating and injecting crafted network traffic into a network that can later be consumed by other devices in a controlled fashion. This is usually done to assess the performance of networks, infrastructure, security controls, and other performance-dependent network tests [1]. They can be implemented over both hardware and software platforms. Hardware platforms are more accurate and can achieve better performance, but they are expensive, closed source, and implemented on dedicated high-performance hardware [4]. Furthermore, this kind of traffic generation tool is generally proprietary and preconfigured to carry out a specific type of tests, making them difficult to customize [2]. Whereas software-based generators are usually open-source and cheaper while being slower and less accurate [1, 4]. These tools are the most wildly used in the networking field mainly due to their high flexibility and open-source nature that allows for easy modifications and extension based on specific research goals [1, 5]. In the last few years, a large number of software-based traffic generators have been proposed in the literature based on different methodologies [2], and most of them were adapted to the current need of network environments [4]. Despite the promising features provided by these tools, the results obtained by experiments are strictly dependent on the ability of the generators themselves to accurately emulate or replicate the network traffic pattern as it is requested by the operator [2, 4]. Thus, in this paper, we propose a quantitative evaluation of the most used open-source NTGs in literature, namely Cisco TRex, Ostinato and Genesids. In particular, this paper, proposes a quantitative comparison of rate from a virtualised traffic generator to assess the performance, scalability, and suitability of that NTG for replaying, or generation malicious traffic for testing, especially in security applications.

The remainder of this paper is organised as follows. Section 2 provides an overview of some prior work that is similar to our work herein. Section 3 presents the chosen network traffic generators Cisco TRex, Ostinato and Genesids, along with a high-level comparison between these three tools. In section 4, we present our testing methodology, the metrics used and an analysis of the obtained results. Finally, section 5 concludes the paper and outlines future work.

2. Related Work

In recent years, many research works have been studied in regard to the performance and precision of the software packet generator, for different purposes [6-8]. In [6], J. Zhang et al. introduced the first classification of network traffic generation that categorises the existing methods for traffic generation into three different methods: traffic generation based on network traffic model; traffic generation based on traffic characteristics and traffic generation based on application protocol. In addition, the paper reviewed the advantages and disadvantages of each category. In another work, Kolahi et al. [7] proposed a quantitative comparison between four network traffic generators, namely Iperf, Netperf, D-ITG and IP Traffic. The comparison was done based

on the throughput and payload size metrics. The experiments were conducted on both Windows and Linux operating systems. This study found that switching the performance monitoring tool used to collect data can make a quantifiable difference in the measured data throughput. Also, they found that most performance tools are designed to run best on Unix/Linux platforms, a factor that is important when considering the operating system of our packet generation system. In previous work, Avallone et al. [9] compared their NTG product with Mtools, Rude & Crude, MGEN, Iperf and UDP Generator. In this work, the experiments were performed on a Linux machine and only monitored the bandwidth of the link using the UDP protocol. Another interesting study by Molnar et al. [2] proposed a common set of metrics that can be used to facilitate the evaluation and also the comparison of different traffic generators. The proposed method divided the studied metrics into five main categories, which are packet-based metrics, flow-based metrics, scaling characteristics and QoS/QoE related metrics. This study found that most of the research work proposed in this context lack any kind of quantitative comparisons and some relevant aspects of traffic characteristics (e.g., multi-scaling properties) are not considered in the evaluation of recent NTGs. The paper claimed that it is important that to validate the packet streams which are being produced by the traffic generators we choose, by capturing the packet streams with a packet sniffer such as Wireshark [10] or TCPdump [11]. This will increase the certainty that the chosen NTG will consistently generate expected traffic, and also help to check for malformed packets that could impact the validity of the testing.

In [12], Horn et al. proposed an empirical comparison of four different implementations of NTGs that allowed for continuous generation of self-similar time series. Fundamentally this comparison focused on the accuracy of data and variance in results, which is not the metric we are using to compare our traffic generators on, but rather the rate of packets transmitted. However, we can apply the principles of repeatability and accuracy informed by the statistical analysis provided by the paper to our testing methodology. In a similar context, Emmerich et al. [13] compared the suitability of different network traffic generators for network testing, by making specific comparisons between hardware, and the impact of CBR traffic on different CPU architectures and packet generators. They also investigated the impact of different I/O frameworks on the performance of the studied NTGs. Another work by A. Botta et al. [9] has analysed the performance of the most used packet-level traffic generators in literature, MGEN, RUDE/CRUDE, TG and D-ITG. This study has pointed out the lack of accuracy of contemporary traffic generators-comparing the throughput of traffic generators such as Iperf [14], Mgen [15] and Rudecrude [16] to a calculated expected value. This is a great comparison, as it provides a frame of reference for the expected values of the traffic generators tested. The testing was conducted using a physical machine and Gigabit Ethernet connection, so, therefore, does not account for the difference in the performance of a virtualised solution.

In this review, we have found that there is no contemporary literature that covers the use cases for which the testing is done. Specifically, this is a quantitative comparison of rate from a virtualised traffic generator. This is to assess the performance, scalability, and suitability of the product for combination with the replay, or generation of malicious traffic for testing. Therefore, we can bring together aspects of previous work cited in this section to inform and improve our methods of testing.

3. Traffic generation tools

Network traffic generators are used to create and then transmit crafted network traffic into a network, that can then be utilised by other devices to test scenarios such as IDS (Intrusion Detection Systems) alert generation and malicious network traffic simulation [17]. They make use of a physical, high-level address and function, for all intents and purposes, as another device on the network. When set up, the traffic generator will either connect to the network and establish itself as a device or utilise the host that it is installed on, virtual or otherwise, to interact with the network on the same network interface. It will then proceed to create newly generated packets, targeted at preconfigured real or virtual devices on the network. Many traffic generators can also use the gateway to route packets so that it appears as though multiple other devices are connected to it and interacting with real devices on the network. The ensuing traffic generated is highly configurable and can be used to simulate many different network usage scenarios, for example:

- Testing if a network can handle the implementation of a new, network-intensive application.
- Stress testing a network, to see whether it can withstand DOS attacks from internal and external sources.
- Testing IDS Systems with threat signatures coming from within the network.

The behaviour of the traffic generated, what layer it functions on, the configuration of different

payloads, headers and flags etc. varies from product to product. Some can be based on packet captures of traffic from the real world, whereas others will craft bespoke streams of packets, or alternatively use random data built using an algorithm in the generator.

latency and minimum CPU resource. TRex is a good stress testing tool and can be used for testing firewalls, load balancing and other load-based tasks as it can concurrently construct packets in multiple streams [18]. However, it does not support routing emulation, making its usage for simulating multiple devices, without multiple instances, difficult.

Table 1. Comparison of the tested NTGs

Packet Generator	State	License	Interface	Customisability	Customisability
Cisco TRex	Stateful/ Stateless	Apache	Command Line / (Community GUI)	Fairly customisable, you have the ability to specify every aspect of the packets generated and build your own stream templates.	Can require more dedicated resources or dedicated hardware to work effectively and adaptably
Ostinato	Stateless	GPL-3.0	GUI	Extremely customisable, supports user defined scripting and total modification of the packet's attributes, e.g., contents, headers and protocols. As well as stream rate, testing method and continuation.	Not suitable for stateful implementations or state reliant traffic.
Genesids	Stateful	OGPL-3.0	Command Line	Moderately customisable. The packet's contents are generated according to a provided list of Snort rules. There are also options for rate limiting.	No direct control of more granular packet settings, e.g., flags.

In order to create and demonstrate realistic attacks on the Cyber-Trust networks the following solutions were considered and tested. Table 1. shows a high-level comparison between the three network traffic generators and gives an overview of the different parameters can be assembled.

3.1. Cisco TRex

The TRex [18] traffic generator is an open-source, flexible, traffic generator that can be used in a stateless and stateful configuration. When being used statelessly, it is possible to craft multiple packet generation streams and alter any field within the packet, including headers, trailers, payload, flags-even sending deliberately corrupt packets to test the network's response. These settings are passed through an algorithm which crafts the packets, assigns them a packet ID and transmits them across the network. It can scale up to 200-400 Gbps,160 MPPS and millions of flows using one Cisco UCS (Unified Computing System), or any COTS (commercial off-the-shelf) server [17], with accurate

3.2. Ostinato

Ostinato [19], often referred as a "Reverse Wireshark" application, is a similar, popular network traffic and packet generator with a friendly GUI. It runs on different platforms including Windows, Linux, BSD and Mac OS X. Like with TRex, Ostinato helps users to create their own packet generation streams; they can also configure them in an in-depth manner, with considerations towards stream rates, bursts, and the number of packets generated [19]. This is a tool suited to network load testing and functionality testing. It also has support for an extremely wide array of protocols, to allow you to tailor packets and stack protocols however you wish. This can be extremely useful for testing edge-case errors while still having extremely high-performance. Ostinato is stateless and as such does not support connection-oriented setups, e.g., a headless browser interaction with a website, which can make it limited for security testing.

3.2. Genesids

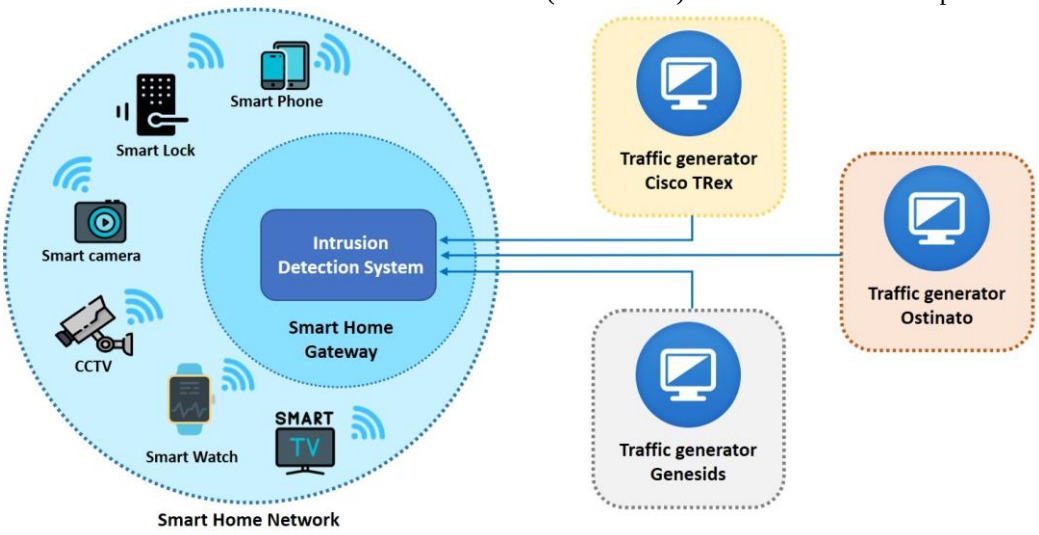

Figure 1. High-level overview of the Testbed

Genesids [20] is a traffic generator specifically designed for high-volume, tailored, security testing of IDS systems. Often to achieve this task, real-world captured traffic is used and replayed across the network. However, this is neither adaptable or suitable for use on a large scale. Genesids allows the specification and generation of application layer payloads, using definitions from Snort rules to craft its traffic. It is ideal for use alongside another traffic generator in order for it to produce realistic background traffic and to maintain the typical load pattern of a network. due to this it is able to reliably produce a large variety of malicious traffic repeatably.

4. Testing Methodology

The testing approach has been proposed in the context of the Cyber-Trust project (https://cyber-trust.eu) in order to evaluate the attack coverage by the Cyber-Trust components and their ability to identify attacks precisely. The traffic generators are used to create an adaptable, realistic attack scenario that can be easily repeated and adapted in a controlled way to allow for thorough and repeatable testing of the Cyber-Trust components.

The Cyber-Trust network, shown in Figure 1. consists of NIDS systems running deep and machine learning techniques on Game-theoretic framework in order to provide recommendations for IoT service providers based on the information regarding vulnerabilities. The premise of the test for each generator was to generate and transmit as many TCP packets as possible within an allocated time period of one minute. This would provide an objective metric that shows the capability and receptiveness to scale of each solution. Each generator was deployed inside a virtual environment that running 2 processors (CPU cores) with an execution cap of 100% and nested paging enabled, with 4GB of RAM and 10GB of disk space (located in a fixed VDI). While the tests have been conducted in a virtualised environment, which will inevitably lower performance compared to hardware-based network infrastructure, the purpose of this testing was to benchmark the performance and suitability of each generator for deployment and use in the Cyber-Trust network for IDS and malicious network traffic testing, which is located within a virtualised network.

To provide as fair of a comparison as possible, each Network traffic generator was set to permit the maximum transmission rate allowed by the virtualised hardware. As TRex is by default almost exclusively hardware limited, the only modifications to the TRex configuration were in the default TCP traffic template by changing the distribution to "seq" (sequential) and the "tcp_ageing" factor to 1. In Ostinato, removing the usual software limit of the host running drone, to generate traffic at the maximum hardware allowed speed, is done by configuring a stream with "packets/Sec" set to 0 in "continuous" mode.

As Genesids focus is customisable malicious traffic generation, rather than capacity, it does not enforce rate limiting by default, only as an option, and as such did not require adjustment. As Genesids does not provide the ability to specify a packet size, rather generating a specific type of packet according to provided Snort rules, the packet size was kept default for each solution.

4.1. Software Unique Configuration

It is worth noting that software specific configuration changes were made from the default settings. TRex was left mostly default, however

Ostinato requires specification of the layer 1-4 traffic, which was as follows: L1: Mac, L2: Ethernet II, L3: IPv4, L4: TCP and no special signatures, flags, or override headers. Genesids requires a list of Snort rules to generate packets from. The rules used in testing were the Snort3 Community Rules v3.0.

It is also worth noting that, despite being under TRex's recommended usage specification, the low-performance, and low-footprint modes [21] have not been enabled in our testing.

4.2. Data Collection

The data for each test was collected by utilizing TCPdump to generate a PCAP file from the captured traffic, allowing viewing of the generated packets from the host. These were captured with the purpose of analysing the packet contents, in order to validate that the payloads had been set correctly, and to check for malformed or corrupt packets during traffic generation.

The resource usage of each NTG was collected using a short bash script, writing the output of top [22] specified for the process ID of the NTG to a file. This allows us to collect the CPU and RAM usage at intervals of one second, over the period of one minute- matching our data collection. This was done to provide accurate system monitoring, without eating into the resources available to the virtual machine.

5. Analysis of Results and Discussion

5.1. Cisco TRex

The Figures 2 and 3. show the resource usage of TRex. Observing the network traffic produced by TRex, the general trend shows a slow start, with the rate of packet generation increasing, then increasing to a baseline of 10929 Packets/s at 8 seconds, before increasing rapidly to 104180 between the 9 and 10 second mark. The rate then becomes less variable, hovering around 110000 Packets/s, with two noticeable spikes to 140000 Packets/s occurring consecutively at the 50 and 53 second marks, where the rate peaks at 140141 Packets/s before normalising. This yields an average rate of 98810 Packets/s generated and transmitted, with a standard deviation of 37526 Packets/s.

As shown in Figure 2., the CPU usage of TRex is high, with the lowest utilisation of 87.2%, occurring at the 500KB/s generation test. Then, the usage continually increases by around 1-2% per 500KB/s, with small but consistent drops in utilisation occurring over the minute period. It is worth noting that when TRex utilises 100% of the first CPU core, it immediately begins using the second CPU core- which is represented on the graph as the points at over 100\% CPU use.

However, the memory usage (see Figure 3.) shows a consistent pattern. The utilisation remains almost entirely static for the duration of each test- increasing by around 0.18\% per 500KB increase in throughput. This is a fairly linear increase in memory usage- with the utilisation staying mostly constant, apart from one jump from 3.0% to 3.1% at the 58 second mark.

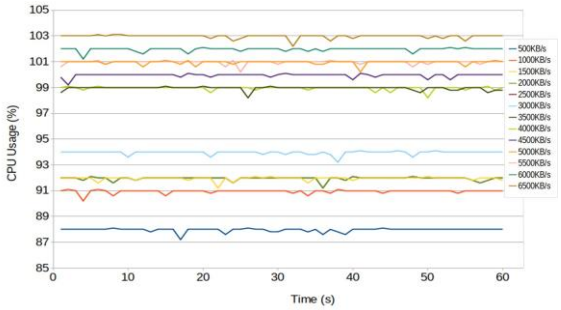

Figure 2. CPU usage of TRex at different bandwidths

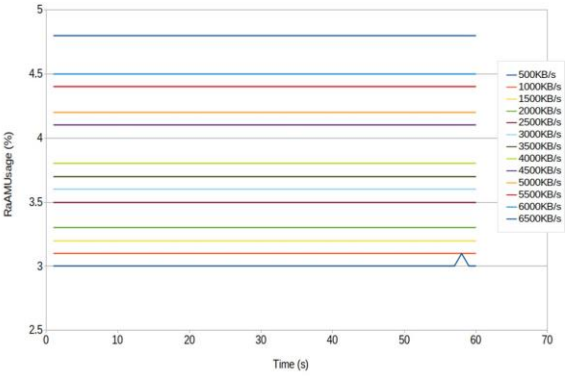

Figure 3. CPU usage of TRex at different bandwidths

5.2. Ostinato

In comparison, the Ostinato packet generation starts much higher, at 121369 Packets/s, with a sharp increase to 167818 at the 5 second mark, before rapidly decreasing again. There is a consistent, large and often extreme variance within this rate, most notably in the period between 18 and 25 seconds, during which the rate drops from the peak at 188608 Packets/s to its second lowest at 89675, a drop of 47.55%. Ultimately, this data shows an average rate of 127271 Packets/s generated and transmitted, with a standard deviation of 23887 Packets/s. Ostinato's CPU utilisation (see Figure 4.) increases by around 2-3% with each increase of 500KB/s generated. There are small but frequent dips, varying from 0.5to up to 2% in utilisation, occurring at varying time intervals throughout each usage test. The increase in CPU utilisation per 500KB/s load is, just like TRex, extremely regular- with increases in the range of 2% and 4% for each increase. Similarly to TRex, the RAM usage for each test, shown in Figure 5, is

extremely consistent, with very little variation occurring over the minute period tested for each throughput value tested. Increases in memory usage were observed on throughput increases from 3000KB/s to 3500KB/s, 4500KB/s to 5000KB/s and 6000KB/s to 6500KB/s. This low memory footprint is a contributing factor to Ostinato's generators good performance in our low-resource virtualised testing environment, with a maximum memory usage of just 0.6% during Ostinato's highest throughput tested- 7500KB/s.

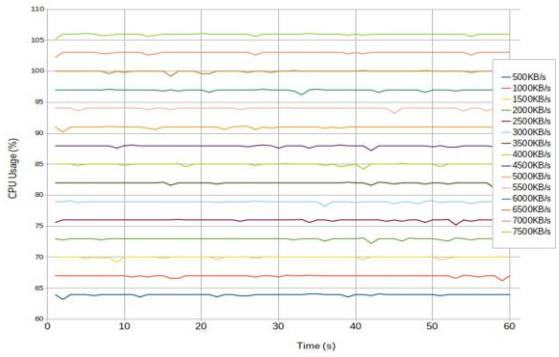

Figure 4. CPU usage of Ostinato at different bandwidths

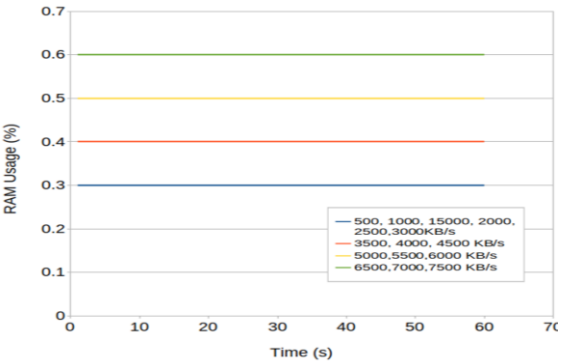

Figure 5. CPU usage of Ostinato at different bandwidths

5.3. Genesids

Genesids provides a much lower but much more stable rate of packet generation and transmission. The rate starts at 11552 Packets/s and the overall trend stays consistent around this baseline, with the average rate of being 11021 Packets/s, with a standard deviation of 844 Packets/s.

As illustrated in Figure 6., in contrast to the other tests, who both used 100% of the first core available, Genesids has a much lower overall CPU usage, peaking at just 29.1% utilisation. However, it does experience a much more frequent and much larger variance in usage, with a maximum variance of 4.1%. It differs in its memory usage, consuming a much larger percentage of RAM, with the trend increasing the longer each test continues. These results in maximum memory usage of 29\% (see Figure 7.).

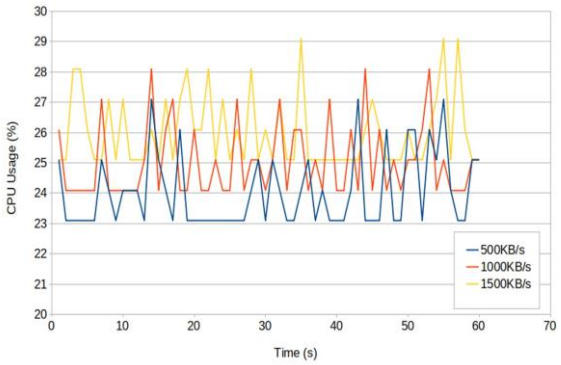

Figure 6. CPU usage of Genesids at different bandwidths

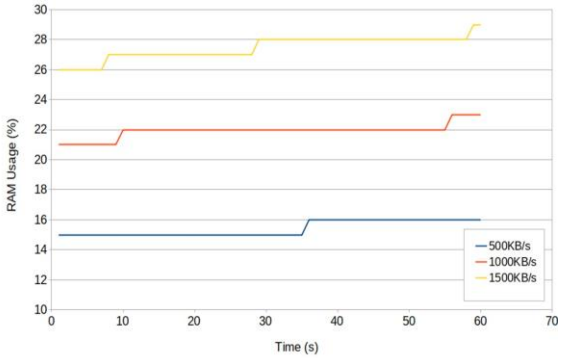

Figure 7. RAM usage of Genesids at different bandwidths

5.4. Rate Analysis

Ultimately our results show that, for our host configuration, Ostinato produced by far the greatest level of traffic, producing on average 28460 Packets/s than the network traffic generator with the second greatest rate of packet generation and transmission, TRex (see Figure 8.).

Furthermore, while comparing the data for the 1-minute time allocation of testing, it shows that Genesids had the least variance, at 7.65%, followed by Ostinato at 18.77% with TRex having the most variance at 37.98%. However, TRex's standard deviation is not representative of the mostly stable trend observed. Calculating TRex's standard deviation after 9 seconds, which excludes the time it takes TRex's rate to reach stability from the beginning of the test, the variance is brought down to 9.95%.

This is a notable difference, as it brings Trex's variance to half of Ostinato's, and second overall. The weighting of this metric is debatable- as only in niche circumstances and edge case testing will there not be time afforded to the traffic generator to reach

its full transmission throughput. However, the stability and predictability of the traffic generated can be an important consideration when load and security testing of networks.

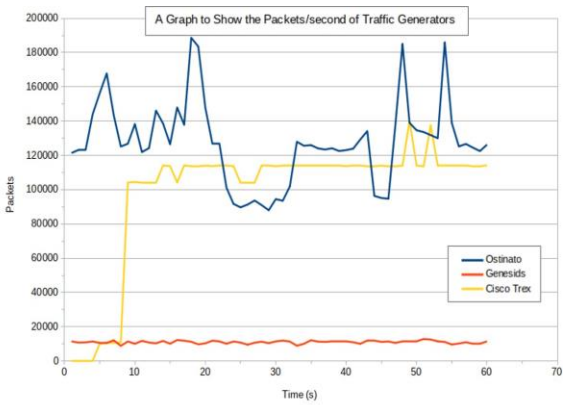

Figure 8. Rate Comparison of the Compared Traffic Generators

5.5. Throughput Analysis

We can also compare the throughput of the traffic generators we tested, this will provide an indication of the amount of raw data that each traffic generator is able to generate and transmit in the time period specified, in our virtualised configuration. In order to do this, we first need to track and compare the average packet size that each traffic generator is producing. We do this we ingress the traffic data into Wireshark, and independently view the capture statistics- this will method for each capture provides us with a unique metric for each packet per second, which are as follows:

- Cisco TRex Average Packet Size: 104 Bytes
- Ostinato Average Packet Size: 60 Bytes
- Genesids Average Packet Size: 140 Bytes

Therefore, we can take these measurements and apply them to our previously collected average rate statistics to calculate the average throughput of each traffic generator. This will allow us to map, and possible explain some of the differences between the traffic generator's approach to generating traffic. We can do this with the following equation:

$$APS(B) \times AR(Packets/s) = AT(B/S)$$

Where APS is the Average Packet Size in Bytes, APR is the Average Rate in Packets/s and the resulting AT is the Average Throughput in Bytes/s. Following this calculation allows us to calculate the following (Figure 9.). This shows an interesting result. Cisco TRex has an average throughput of 10035 KB/s with the average rate of 98810 Packets/s that was recorded during the testing; comparing this result to the traffic generator with the highest rate Ostinato, who's average throughput was actually lower, at 7457 KB/s with the average rate of 127271 Packets/s. This is a difference of 2578 Packets/s, or 26%. This is due to the difference in average packet size that we explored previously, despite Ostinato's lead in terms of rate- its lower packet size does show that it does not have the highest traffic throughput of the traffic generators tested.

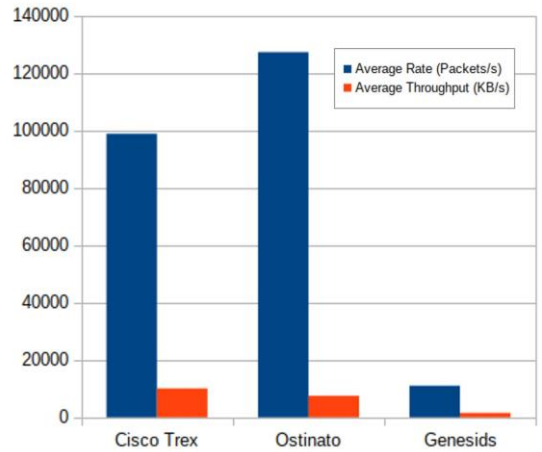

Figure 9. Rate and Throughput per Traffic Generator

Furthermore, if we are comparing the ratios of average rate to average throughput, we see that Genesids- while having the slowest rate, does have the highest throughput to rate ratio. For our test case these factors are secondary to the rate of packets generated, however in some workflows or scenarios of testing- the throughput of data could be a key factor that needs to be isolated, monitored or controlled, therefore it is important for us to take account of these metrics.

Furthermore, use cases where the throughput of data is the test metric, for example- stress testing of network connections under much greater load, e.g., during a DDOS attack, this aspect of the traffic generators can be key.

5.6. Reaching Maximum Throughput

A solution that often may be demanded by researchers is to simultaneously load test a network, while also injecting generated malicious packets into the network. This is a use case that allows for the testing of a network load- simulating benign traffic at low, medium and high throughput and stress on network components as well as testing the response of security controls e.g., firewalls, IDS systems and gateways. Together- this allows a researcher or network administrator to create a precise, repeatable and customisable network test that can map and test the network devices responses to malicious attacks under normal load- and the impact that said mitigation can have on normal network functionality.

Exploring a theoretical test network- that would use a standard network throughput size of 10GB/s, a throughput level that most small to mid-level networks will handle at peak and a baseline background traffic level that some medium to large networks will have constant load of.

In order to achieve this 10GB/s load and achieve maximum throughput for a 10GB/s NIC- we propose that a solution utilising a combination of traffic generators- combining facets of each to create a single, multi-purpose solution. This solution would allow for high-volume benign traffic to be generated by one traffic generator such as Ostinato or TRex, with the customisable malicious packet generation capabilities of Genesids for example. Leveraged with higher resources than the virtualised scenario we explored- or conducted from dedicated hardware, could meet the network throughput goals outlined and reach the maximum throughput of a 10GB/s NIC.

5.7. Evaluation

Although these results do not definitively prove that Ostinato is the best packet generator out of the three compare traffic generators. It does provide an objective metric to state that, for this low resource, virtualised configuration- Ostinato performed the best for the metric of highest average Packets/s generated while providing an infrequent, but notable variance of 18.77%. The architecture of each packet generator is likely the explanation for the varied result, as each solution uses a different architecture and generation algorithm. Ostinato uses a unique architecture written in C++ to transmit packets, where Genesids utilises libcurl [23]. TRex uses a different approach, utilising the DPDK Kernel interface [24] to directly interact with the Linux kernel, behaving as a loadable kernel module to interface directly with the kernel network stack. This approach may function comparatively worse on a single-threaded application like the one tested- however it may be more scalable with more allocated processors. This low resource allocation could explain the large divide between TRex, Genesids and Ostinato. For example, The Enterprise Stack, as well as TRex's own hardware recommendations [25] per their installation manual [21], recommend that a virtual machine running TRex should have at least 4GB of Memory and 4 virtual processors. In our testing, we only allocated two. Further testing would be necessary to ascertain how much of a difference this makes.

6. Conclusion

In this paper, we presented our testing methodology of the traffic generators Cisco TRex, Ostinato and Genesids. The implantation of the packet generators is done within the Cyber-Trust network to test the IDS and Machine Learning components to a measurable and repeatable level in a controlled environment. By combining the clean packet generation from the high-performance Ostinato with the customisable malicious packet generation ability of Genesids; we can leverage a robust, scalable performance of both solutions to create adaptable and realistic attack scenarios. The scenario of attack can be easily repeated and adapted in a controlled network environment. Furthermore, we can adapt the malicious and clean packet generation to simulate attacks on the whole network, key devices, or individual devices as well as inside, or outside of the Smart Home network. This will allow for thorough and repeatable testing of components within a virtual environment, all without the need for manual exploitation or attacks of networks which can be costly and time-consuming. This can be invaluable for the research community as a whole when dealing with the limitations, requirements and availability of many different and emerging packet generators that can aid in the collation of needed information. The accuracy of our testing can be evidenced through the consistency of the results we acquired throughout our testing, with an average standard deviation for repeat testing of ±21.29\% for our NTG performance tests. We can also compare our results to that of known benchmarks. Comparing our TRex results with that of the TRex virtual machine installation and testing guide [21] a result of "9.99 Kpkt/sec" is shown, a result nearly mirrored by our average of 9.64Kpkt/sec despite the decreased resources.

While there are still restrictions in hardware requirement, the limitations herein are held against the availability of network hardware resources and can be scaled as seen fit for purposes of your own. As such, the benefits of our testing can be seen. The tests were all conducted in the same environment with constant machine specifications and targets. This resulted in each NTG having the same opportunity to perform to its best, under the deliberately constructed poor conditions. This tested a factor that is not often considered with high-performance, multi-state NTG's- which is their scalability to low-end, often virtualised, use cases. Ultimately, our research and testing could be expanded by comparing the resource usage and performance of the NTG's over a longer time period to explore performance for long-term use.

7. References

[1] L. Angrisani, A. Botta, G. Miele, A. Pescapè, and M. Vadursi, “Experiment-driven modeling of open-source internet traffic generators,”IEEE transactions oninstrumentation and measurement, vol. 63, no. 11, pp.2529–2538, 2014.

[2] S. Molnár, P. Megyesi, and G. Szabó, "How to validate traffic generators?" in 2013 IEEE International Conference on Communications Workshops (ICC). IEEE, 2013, pp. 1340–1344.

[3] M. Paredes-Farrera, M. Fleury, and M. Ghanbari, "Precision and accuracy of network traffic generators for packet-by-packet traffic analysis," in 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. IEEE, 2006, pp. 6–pp.

[4] P. Emmerich, S. Gallenm¨uller, G. Antichi, A. W. Moore, and G. Carle, "Mind the gap-a comparison of software packet generators," in 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 2017, pp. 191–203.

[5] S. Avallone, D. Emma, A. Pescapè, and G. Ventre, "High performance internet traffic generators," The Journal of Supercomputing, vol. 35, no. 1, pp. 5–26, 2006.

[6] J. Zhang, J. Tang, X. Zhang, W. Ouyang, and D. Wang, "A survey of network traffic generation," 2015.

[7] S. S. Kolahi, S. Narayan, D. D. Nguyen, and Y. Sunarto, "Performance monitoring of various network traffic generators,"2011 UkSim 13th International Conference on Computer Modelling and Simulation, 2011.

[8] S. Behal and K. Kumar, "Characterization and comparison of ddos attack tools and traffic generators: A review." IJ Network Security, vol. 19, no. 3, pp. 383–393, 2017.

[9] 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. IEEE, 2004, pp. 316–317.

[10] "Introducing wireshark," Wireshark for Security Professionals, p. 1–18, 2017.

[11] Linux, "tcpdump (8) linux man page," https://linux.die.net/man/8/tcpdump, (Access Date: 16 September, 2020).

[12] G. Horn, A. Kvalbein, J. Blomskøld, and E. Nilsen,"An empirical comparison of generators for self-similar simulated traffic," Performance Evaluation, vol. 64, no. 2, p. 162–190, 2007.

[13] P. Emmerich, S. Gallenmuller, G. Antichi, A. W. Moore, and G. Carle, "Mind the gap - a comparison of software packet generators," 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2017.

[14] V. GUEANT, "iperf - the ultimate speed test tool for tcp, udp and sctptest the limits of your network internet neutrality test," https://iperf.fr/, Access Date: 03 July, 2020.

[15] MGEN, "mgen," https://www.nrl.navy.mil/, (Access Date: 15 August, 2020).

[16] RUDE, "Rude crude," http://rude.sourceforge.net/, (Access Date: 26 September 2020).

[17] F. Erlacher and F. Dressler, "Testing ids using genesids: Realistic mixed traffic generation for ids evaluation," in Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, 2018, pp. 153–155.

[18] "Cisco trex," https://trex-tgn.cisco.com/, (Access Date: 21 September 2020).

[19] "Ostinato," https://ostinato.org/ (Access Date: 17 September 2020).

[20] Felixe, "felixe/idseventgenerator," https://github.com/felixe/idsEventGenerator, (Access Date: 22 September 2020).

[21] KernelNIC, "Trex user documentation," shorturl.at/nDJX5, (Access Date: 5 July, 2020).

[22] A. D. Cahalan, "top (1) - linux man page," https://linux.die.net/man/1/top, (Access Date: 26 September, 2020).

[23] "libcurl - the multiprotocol file transfer library," https://curl.haxx.se/libcurl/, (Access Date: 16 September, 2020).

[24] KernelNIC, "Kernel nic interface," shorturl.at/copuS, (Access Date: 26 September, 2020).

[25] T. E. Stack, "Build and run cisco trex traffic generator," The Enterprise Stack, Sep 2016, http://theenterprisestack.com/2016/09/12/build-andruncisco-trex-traffic-generator/, (Access Date: 22 September, 2020).

8. Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 786698. This work reflects authors' view and Agency is not responsible for any use that may be made of the information it contains.

