Experimental Assessment of Benchmark-oriented Network Traffic Generators

Vasileios Papadopoulos
Department of Telematics Engineering
Universidad Carlos III de Madrid
vassilispapadop@gmail.com

Abstract—The burstiness of network traffic has a profound impact on the performance on many network protocols in areas such as, congestion control (eg. TCP), multiple-access (eg. CSMA/CA), routing (eg. BGP), switching and multiplexing in general. Network measurement performance tools, have been widely used to exploit vulnerabilities, monitoring reports and testing newly, under development protocols. Network Traffic Generators (NTG), are capable of tracing burden from real-life traffic and replicate it offline, generating load under predefined traffic profiles such as, CBR, VBR and traffic according to well known probability distributions (Poisson process and Poissonian arrival). Though, it is not clear how accurately these tools are performing, as little study has been done. A traffic generator is required to satisfy hard real time requirements; time sensitive applications (eg. VoIP, Video), replication/emulation could lead to different than the expected experimental outcome, mainly because of the intrinsic limitations of the PC architecture and NTG’s design. Processes are competing each other for CPU attention, rising up uncertainties possibility and higher kernel overhead. The accuracy of timers, the network socket family and the policy of process scheduling should be consider as main sources of fluctuations and eventually bursty traffic generation. In this work, we use several well known statistical tools for capturing anomalies in frame generation process. Developers should take into account the rising limitations for better NTG design and implementation.

Keywords—Network traffic generators, iperf, mgen, c/rude, d-itg, burstiness, accuracy.

I. INTRODUCTION

The goal of this work is to present a framework capable of reporting NTG’s capabilities, performance, limitations and design strategies. We define traffic generation to result in timestamped series of packets arriving at and departing from a particular network interface with realistic values for at least the layer 3 (IP) and layer 4 (TCP/UDP) headers as well as the whole ethernet frame (Preamble+Payload+Headers+IGP). This traffic should accurately reflect arrival rates and variances across a range of time scales, e.g capturing both average bandwidth and burstiness. Most of traffic generators, report average achievable bandwidth in relatively large time scales hiding their inadequate to provide deterministic generation process or other user defined network traffic profiles. In this work we are focusing on possible bursty characteristics of birth/generation process originated from four widely used traffic generators. Information regarding to NTGs are depicted to Table I. Our hypothesis is that realistic, responsive and deterministic packet generation must be at least informed by the hardware, software and protocols hosting the programs (NTG).

A. Why accurate generation is critical?

Protocol designers, network administrators and whom ever be concern, they all willing to generate network traffic under pre defined profiles for performance evaluation, maintenance and protocol benchmarking etc. Testing tools have been progressing over the years aiming to improve their functionalities, meeting quality guarantees at the highest level. From protocol designers perspective such tools should be working just perfectly. Though, some factors could affect the performance of tools such underneath platform, architecture design and eventually could lead to differently than the expected results. Software designers, aiming to produce applications having always extensibility in theirs user-requirements features. All softwares nowadays are cross-platform. Although, for achieving this, applications are designed to run at the application layer increasing their interaction with underneath platform. That user-space to kernel-space interactions are needed for critical operations, adding negligible overhead though. The most widely used generators are running at user-space. Thus, not only their capability and design strategy but also system kernel flexibility and performance could form a different than the expected experiment outcome.

Link burstiness can be considered as the operation region outside of the 95th percentile quality of service guarantee rule. But could greatly affect protocol performance. For example, MAC-layer immediate retransmissions are much less effective on links with bursty traffic and eventually bursts of losses than on links with independent losses. This means that two networks can have identical connectivity and packet reception ratio, yet exhibit completely different performance. Burstiness characterization will help to understand why some protocols behave differently on similar networks and will provide insights into tuning protocol parameters to improve performance. Though, all the analysis relies on the assumption that the generation

<table>
<thead>
<tr>
<th>Table I. Traffic generators considered in the assessment.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Written in</td>
</tr>
<tr>
<td>Licence</td>
</tr>
<tr>
<td>Citations</td>
</tr>
<tr>
<td>Platforms</td>
</tr>
</tbody>
</table>
process doesn’t introduce by itself any bursty characteristics. As we will see later on this work, this is a weak assumption and more than often someone would think, NTGs add background noise to frame generation process.

B. Related work

In work at [1], authors tried to address the similar question as our work. They perform a comparative analysis in terms of achievable throughput and variance among several runs of the same experimental scenario. In [2], they perform packet-by-packet analysis and focusing on how the real-time kernel extensions could increase granularity of software timing requirements. They are also concerned of possible bursty traffic generation, although make use only coefficient-of-variance technique to capture such traffic. Additionally, authors in [3] explore any correlation in time interval between two consecutive packets. Though, they are focusing on the inter-departure time as this observed at layer 3 (PF_FILTER).

C. Our Contribution

The question of how accurate your traffic generator tool has been addressed before, the studies lacked from measurement accuracy. Although, a fully test-bed description will follow in next sections, it is worth mentioning that our work relies on dedicated hardware with nanosecond packet timestamping process accuracy. The sniffer is capable to introduce low, deterministic processing delay when frames arriving to its ports. Additionally, we make use of several well know statistical tools for discriminating patterns in generation process and possible bursty behaviour.

The rest of this work is organized as follows. At next section will discuss how and when a traffic flow could be not deterministic and we will present some techniques that could help us to capture the notion of bursty traffic flow. Then we present all the architectural limitations and concerns that arising when traffic flow is generated by soft real time systems instead of embedded systems. We present our test-bed and methodology and the main evaluation metrics that we use to characterize flows. Finally, we replicate the laboratory results to a single server queueing system simulator, turning our attention on how the mean system residence time varies from the expected result.

II. NETWORK TRAFFIC

Understanding network traffic behavior is essential for all aspects of network engineering and operation. Component and protocol design, management, modelling and simulation features are all relying on deep understanding of either internet or local traffic. In this section, we will address the concept of bursty behavior as this observed in ethernet single domain network topology, as well as methods for identifying and capturing such characteristics.

A. Notion of Burstiness

Traffic measurements and burden characterizations are vital for, the analysis and assessment of network performance and for network design and cost optimization. Packets in some applications are expected to arrive at specified times (VoIP, Video). Although, the nature of NTGs and their underneath platform, could force packets to arrive in clustered manner. This phenomenon is known as packet burst. Burst , is a group of consecutive packets with shorter inter-packet gaps than packets arriving before or after the burst of packets. Burstiness can be harmful to any traffic flow analysis of network assessment.

- Definition: A traffic process A is said to be more bursty than a traffic process B if a single server queuing system yields a longer system size tail distribution when the process A is used to drive the queueing system.
- Queue size normalized by service rate + service time = delay time.
- When buffer size is finite, system size tail distribution estimates packet loss probability.

B. Capture Techniques

There are a lot of techniques out there that are used to define and measure burstiness. Some of them, provide limited information about the flow and some are more complex. Peak-to-average ratio, is a simple measure to represent traffic burstiness. It provides a ratio of peak packet rate to the average packet rate over some specified averaging interval. It does not describe tendency in network traffic. Coefficient-of-variance (C.O.V), measures variability in relation to the mean. The C.O.V is equal to zero for deterministic arrivals and one for Poisson arrivals, while bursty traffic has a C.O.V greater than one. Table II depicts VMR for well known distributions. Although this approach is applicable for measuring the alterations in the data traffic it cannot capture the correlation between consecutive packets. The correlation in the measured traffic can be captured by calculating the variance of the time between n arrivals and comparing it to n times the variance of successive arrivals. Indexes-of-dispersion also known as Fano Factor is a windowed analysis of variance normalized by the mean in particular interval. Finally, we could sense the notion of burstiness of a flow based on its behavior through a network element, specifically on the queue size behaviour and average residence time.

III. ARCHITECTURAL LIMITATIONS

In general, time-stamping packets is tricky. For any traffic analysis which requires high precision a way to accurately mark packets is far from a simple requisite. The software-only implementation, timestamps packets in the application layer which is furthest away from the physical layer. This implementation yields the least accuracy due to the largest amount of delay and jitter that occurs in passing the time-stamp through the various layers. Software timestamps typically give errors on the order of microseconds to milliseconds depending
on the operating system and platform. Such a timing mechanism provides no guarantees. Next we will discuss key factors that could yield to inaccurate time-stamp mechanism as well as some architectural limitations that play significant role to software timing requirements.

A. Scheduling

In multi-user environments, tasks and processes competing for CPU attention. Catalytic role to resource assignment is played by an OS module, called scheduler. A scheduler, is liable to distribute CPU cycles among variety of tasks that concurrently have to be executed, according to a scheduling algorithm. There are number of algorithms for scheduling tasks on a single processor; some of them are used for scheduling tasks on multiprocessor system either under the partitioning scheme or under the global scheduling scheme. A partitioning scheme adds the constraint that all instances of a task must execute on the same processor. When scheduling hard real-time tasks on a multiprocessor system, it is important to ensure that no task admitted in the system misses a deadline. This implies that a stringent admission control is required for hard real-time systems [3]. A real-time system is one in which tasks that must functionally results in a timely manner. This connotes that the tasks submitted to the system have known timing necessities. These tasks are called real-time tasks. In general, RT 1 kernel extensions are used to integrate better to applications with strictly timing requirements. The OS scheduler is loaded as a process and could be preempted by other high priority task. This functionality could decrease system-kernel overhead. A NTG, is a software system consisting of several strictly real-time tasks. However, they rely on scheduling policies and limitations of each particular platform. Kernel has to react and make decisions immediately, without adding extra queue waiting time to currently active/ready processes. The scheduler is concerned essentially with:

- Throughput - The total number of processes that complete their execution per time unit.
- Turnaround time - Is the difference of time between the time of arrival and time of dispatch/completion of a process.
- Response time - amount of time it takes from when a request was submitted until the first response is produced.
- Fairness - Equal CPU time to each process.

In practice, no matter which scheduling policy a kernel is deploying, it is hard to meet service guarantees for strict real-time applications in soft real-time environments. Although, some policies can favourite processes which are having timing constraints. In our concern, this translates into time offset of frame arrival from its theoretical point. A lot of experimental analysis has been performed relying on soft-time platforms, among several networking disciplines. Even though, the NTG is running as the process with the highest priority in the system, has to take CPU cycles/time for its operations. NTG will suffer by OS event scheduling accuracy and probably would face de-scheduling; the testing tool makes an OS system call to some form of textttsleep or time-slice expiration. In fact, traffic generators has to give up by cpu contention process every time the next sending packet that has to be addressed it would happen according to user-defined time between two consecutive packets.

B. Socket Family

The idea of a socket is to plug-in to the other process and send the message, with the operating system taking care of delivering it. The socket layer supports communication between processes that are running, either on the same machine or different machines. Sockets can be opened using a number of different protocols including the Unix socket for intra-machine communication and TCP sockets for inter-machine communication. The operating system provides a set of primitives that enables the computer to communicate with other machines using standard protocols such as Internet Protocol (IP), Novell IPX, AppleTalk, and Microsoft SMB. These higher-level protocols are, in turn, built upon Ethernet (IEEE 802.3) or other Media Access Protocols. The operating system primitives include device drivers for the communication hardware, packet routing and filtering functions, and implementation of the protocol stacks such as Transmission Control Protocol/Internet Protocol (TCP/IP). Typically, for applications which have not tight timing requirements, the choice of selecting socket family is based on functional criteria. Although, a NTG has to choose a family that guarantees the lowest time overhead and performance. The authors at [5], have analyzed two commonly used families of sockets available in Linux kernel 2.4: the INET and the PACKET socket families. They have measured the cpu-cycle-latency and found that, the PACKET family approach can reduce CPU cycles up to 50% compared to INET. PACKET family expect a complete Ethernet frame that is created at user-space layer. On the other hand, the INET socket family performs the set of operations required to construct UDP/IP and MAC headers.

C. Timers Accuracy

The system clock plays an important role in resource management, particularly in real-time applications. The clock allows applications and the kernel to query the current time, which is crucial for dealing with time constraints. It also provides an alarm clock, which the operating system and applications use to be notified at specified times. High performance timing is tricky. Timing is one of those things where a little knowledge can be problematic. A common way for doing timing is to use functions like gettimeofday(). Such functions return what is called a wall time, which is the time that corresponds to a calendar date/time. These clocks suffer from some limitations. They have low resolution. A NTG can not rely its high performance timing operations on low granularity. Furthermore, the clock can jump forwards and backwards in time causing a time drift. Some systems have NTP 2 enabled which periodically adjusts the system clock to keep it in sync with actual time. A different approach is to use system clock. The application has to access the clock value and the rate at which it increments. The increment rate is the cpu frequency. Subtracting two successive clock reads and dividing the result to clock frequency, will give as the floating point interval in seconds. A more complex approach to provide high accuracy

\[2\text{Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over data networks.}\]
timing guarantees to applications with high time sensitivity, is the use of Read Time-Stamp Counter (RDTSC). RDTSC provides access to the current value of the processors time-stamp counter to upper layers. Although, several concerns are arising. The counter is processor core specific, meaning that using RDTSC will give different values on different processor cores. This causes non-monotonic clock behavior as a thread is migrated across cores during execution. It doesn't always tick at the same rate, as the clock frequency will vary as the CPU load changes. This due to power saving features in the processor that throttle down the clock speed when the load is low. A convenient way when using RDTSC is to set CPU Affinity to NTG.

D. Is a modern OS able to keep up the rate?

Network drivers are far from the simplest drivers in the kernel. They push the speed boundaries of modern multi-core processors. Almost every new system comes with a 1Gigabit per second (Gbps) network card. Taking into account that the smallest Ethernet frame size is 64 bytes (plus 8 bytes for synchronization and a 12 byte inter-frame gap), a 1 Gbps network card should (by specification) be able to handle:

\[ 1 \text{Gbps/672} \text{bits/frames} = 1.488 \text{M frames/second}. \]

This leaves us with a processing time of about 672 nanoseconds per frame. A 2 Gigahertz processor can execute about 1,400 cycles in that time and a single instruction can take multiple cycles. Is a modern operating system able to keep up with that rate? The simple answer is: no. The standard Linux kernel can't keep that pace for an extended period of time as it relies on buffering to handle short bursts of traffic beyond what it can handle. It still does its best to get as many packets as possible using modern network card resources and the multi-core processors available on the machine.

IV. TEST-BED AND METHODOLOGY

Even though our work is intended to be applied to every network deployment, such as wired or wireless infrastructures, we have been following the simplest way. The goal of our methodology is to fulfill the following principle: Experiments must be repeatable and to guarantee reproducibility of the results. For that reason, we decide to choose a single domain wired network (LAN) instead of an isolated wireless network (WLAN), mainly because a radio environment makes repeatable experiments harder to achieve. The impact of stochastic factors is closely bound to the variance of the outcome. Medium contention parameters in typical 802.11 environment, are used for achieving network stability and fairness among stations. Though, this some sort of randomness could harm the main idea behind of this work. Furthermore, as we will see later on this paper, our accuracy granularity reaches nanoseconds resolution, which makes a wired network deployment the most suitable for our needs. The figure 1, depicts our laboratory test-bed. Two identical PCs are powered up by Intels Core 2 Quad CPU, clocking @ 2.66 GHz with 4 GB of RAM. Two wired cards, namely, one integrated Intel 82567 Gigabit Ethernet card, and one PCI RealTek, are interconnect the two machines.

The accuracy of the measurement tool is a critical factor when performing experimental work figure 2. The work in [4] has showed that any kind of network traffic sniffing tool should not be trusted. Commonly used sniffing software, tshark relies on libpcap and PF_FILTER for capturing packets. Although, tshark provides plenty of features for network analysis, lacks of timestamp accuracy and precision. For the reasons described above, a good choice was to install to our test-bed a network tap \[ ^3 \] [11]. A network tap is able to capture traffic in high speed networks with extremely high time-stamping process accuracy.

Our measurement methodology, based on the analysis of the time interval between to consecutive frame arrivals, as this observed by the sniffer. The packet generator machine, injects traffic to the link with fixed frame length and constant bit rates. The table III depicts the different use cases we used for our analysis and evaluation approaches. The same experimental scenario was repeated 5 times and the duration of each run is 30 seconds.

A. Considered Network Traffic Generators

1) IPERF: IPERF [6] is a bandwidth measurement tool which is used to measure the end-to-end achievable bandwidth

\[ ^3 \text{NetOptics tap and the Endace DAG card} \]
by using TCP/UDP streams. It allows tunings in parameters like TCP window size and number of parallel streams. End-to-end achievable bandwidth is the bandwidth at which an application in one end-host can send data to an application in the other end-host. IPERF approximates the cumulative bandwidth (the total data transferred between the end-hosts over the total transfer period) to the end-to-end achievable bandwidth. IPERF is used widely. The popularity and widespread use can also be partially attributed to its ease of installation and absence of kernel or device driver modifications.

2) **MGEN**: MGEN [7] provides the ability to perform IP network performance tests and measurements using UDP and TCP IP traffic. The application itself is an extension of Protean Protocol Prototyping Library (Protolib), which provides a C++ framework for network protocols development. MGEN, implements traffic control and network management. It is suitable for low power capability systems with huge kernel overhead.

3) **C/RUDE**: Next traffic generator considered in this assessment is C/RUDE [8]. C/RUDE originally designed and developed to outperform some timing limitations that MGEN tends to ignore. MGEN rely its operation on systems timer (see. Architectural Limitations : Timers ) which only provides timer resolution up to milliseconds magnitude.

4) **D-ITG**: D-ITG [9] is the most sophisticated traffic tool. Is a platform capable to produce traffic at packet level accurately replicating appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) random variables (exponential, uniform, cauchy, normal, pareto, etc.)

V. PERFORMANCE ASSESSMENT

In this section, we present the results of tests carried out in a test-bed running Linux kernel 2.6 [12][1]. With these tests we intend to determine the maximal performance of all TGs and evaluate the stability and the reliability of the generated packet flows. Furthermore, we turn our focus on how far NTGs inject packets to the network from the optimal point of frame generation. Section III, presents some common limitations to frame generations and shows, how a bad component choice could turn against in such tools operations. Several well known models are used to characterize as accurate as possible the generators.

The Figure 3 presents the CDF for two use cases. The saturation case for 100B and 1470B payload as well as 90 of maximum achievable bandwidth. The vertical line represents the theoretical interpacket gap that frames should arrive at the other end of the communication link. An important observation is that all traffic generators fail to reach up to theoretical point of packet generation. This it has to deal with Linux’s kernel nature and the overhead that it is introduced by its modules for operations such as scheduling etc. It is intuitively clear from Figures 3 and 4 that we could categorized the NTGs into two different generation policies software-based tools. IPERF and C/RUDE try as much as possible to inject traffic into a more deterministitc manner in contrast to MGEN and D-ITG which tend to deploy generation control mechanisms to...
achieve on larger time scales more accurate injection. They introduce scaling generation management to balance out any possible drift to the birth process. Figures 5 and 6 hint out the different approaches deployed by NTGs. Even though the above analysis shows that all traffic generators losing its accuracy and there is a drift to observed inter-packet gaps at the sniffer side, it is not clear whether there is a correlation or any persistence pattern to frame generation process. Next sections try to tackle this.

A. Auto-correlation

Auto-correlation refer to the correlation of a time series with its own past and future values. Positive auto-correlation might be considered a specific form of persistence, a tendency for a system to remain in the same state from one observation to the next.

\[ R(k) = \sum_i (T_i - T_{ix})(T_{i+k} - T_{ix})/(n-k)\sigma^2 \]

In contrast, negative correlation hints out a tendency for oscillation among different system states. We define the error as the distance of an observed inter-packet gap from the mean and we calculate the auto-correlation with the above equation. We are interesting in capturing consecutive inter-packet gaps whose absolute distance of the probability distribution mean is equal.

B. Dynamic ALLAN variance

In this section we borrow basic concepts and principles from clock stability assessment. We treat NTGs as a clock, whose clock tick rate is the frame generation rate. Ideally, every clock should keep its stationary properties throughout its life period. Well tuned clocks, produce white frequency, white noise.

Non-stationary Clock Analysis: The mean and the variance of the underlying process are not constant and the auto-covariances dont depend only on the time lag. A perfect tuned clock could be able to produce white frequency noise whose standard deviation and mean remain constant over time. If only white noise is present in signal, then the noise will be reduced by the square root of the number of points used for the averages. But, even if exists a low frequency drift in data, then after a certain number of averages the coefficient is shifted because of the drift of the data. The DAVAR is in practice obtained by sliding the estimator of the Allan variance on the data. We also define the dynamic Allan deviation, or DADEV, as the square root of the DAVAR (the DADEV estimator is defined in an identical way). There is a typical

![Fig. 6. Overview of observed inter-arrival time at sniffer versus expected value. Use case : 100B, 90 saturation.](image)

![Fig. 7. Auto-correlation of the errors. Use case : 100B, 90 saturation.](image)

![Fig. 8. Dynamic Allan variance. Use case : 100B, 90 saturation.](image)
tradeoff in the computation of the dynamic Allan variance. If the window is long, the variance of the estimate is small, but the localization of events in time is poor. Conversely, a short window guarantees an excellent localization of events, but has a poor variance reduction. It is better to choose the window on a case-by-case basis, depending on the type of data considered. The dynamic Allan variance can be applied in several different ways. In general one can think of two ways of using DA\textsc{var}:

- To prove that a clock is behaving in a non-stationary way and to identify the types of non-stationarities that are presented.
- To guarantee that a clock is behaving in a stationary way. A clock may in fact fail the requirements locally but meet the global requirements specified by the classical Allan variance. This violation of the requirements can be revealed only by showing the dynamic Allan variance.

C. Burst Length

Our analysis in previous sections have shown that, none of the four traffic generators are capable to produce traffic in deterministic manner. In fact we can conclude that all NTGs can be considered to behave in the same way, when we look in higher time scales. Dynamic Allan variance method give us the intuition that, NTGs meet global requirements but not locally. But, what's the significance of this non deterministic behavior? To address this, we developed an algorithm which runs in Linux shell, for capturing any possible packet trains within the generated process. At this point it is convenient to introduce some of the basic tuning parameters of burst detection algorithm.

- ta - inter-arrival time of packets in burst.
- dta - inter-arrival time tolerance.
- pi - a size of a packet.

It is intuitively clear, tuning parameterization diversity which is consisting of the above factors, could give a completely different burst length distribution and characterized. By setting the inter-arrival time at certain level we differentiate certain number of bursts. The inter-arrival time tolerance variable is used to form up a capture band, a zone in which the algorithm marks packets.

Many burst detection algorithms are based on spike analysis and use a window to integrate a spike train signal and a threshold to detect a burst occurrence. Thus, important tuning parameter for our algorithm, is the process trimming level. At first glance, threshold can be defined as the theoretical arrival rate. Packets tend to arrive to the other end of the communication link in clustered manner. Every observed inter-packet gaps that is below a threshold is candidate for start of train. To indicate the end of a cluster, a frame has to arrive later or equal to the considered threshold. An algorithmic overview of code is presented above. It is worth to mention that our work is based on post-processing analysis. A real-time spike detection software development is at the top of our future work queue.

\[ gap \leftarrow GAPS.first \]

\[ \text{while} \ GAPS \ \text{do} \]

\[ \text{if} \ gap \leq \trim - \text{level} \ \text{then} \]
\[ \text{consecutive} \rightarrow \text{frames} \leftarrow ++ \]
\[ \text{else} \]
\[ \text{if} \ \text{consecutive} \rightarrow \text{frames} \geq 2 \ \text{then} \]
\[ \text{burst} \rightarrow \text{length} \rightarrow \text{frames} \leftarrow ++ \]
\[ \text{end if} \]
\[ \text{if} \ \text{consecutive} \rightarrow \text{frames} \leftarrow 0 \]
\[ \text{end if} \]
\[ \text{gaps} \leftarrow GAPS.next \]
\[ \text{end while} \]

As we can observe from figures 9,10,11,12 traffic generators produce different burst lengths characteristics. Depending the bandwidth level and the size of frame, some generators outperform others by injecting network traffic more accurately and in more deterministic manner. The slope of CDF lines, intuitively point out the distribution tail size of the burst length analysis. An extension to previous approach, is to expand the classic burst algorithm detection by adding functionality in which measure the depth of the burst. From queuing theory perspective, smaller inter-packet gaps leading to different system states, as increasing the average residence time for incoming packets and delay. The packet loss probability de-
Fig. 11. Burst length CDF for 5 measurements. Use case: 1470B, 50.

Fig. 12. Burst length CDF for 5 measurements. Use case: 1470B, 90.

Fig. 13. CPU cycles as measured by PERF tool. Use case: 100B, saturation.

VI. RESOURCE MANAGEMENT

A. Consumption

Availability of system resources is critical for any either user-space or kernel space process. In this section, we preliminarily present base resource consumption as this captured by means of number of cpu cycles that are required for each experiment completion. According to authors knowledge a proper way to measure cpu consumption is to have privileged access to source code, for manipulating it in way that could report consumption at specified part of the code. Though, this is included as part of our future work.

We could distinguish NTGs, as tools that are self-oriented and rely most of the required frame generation and transmission functionality in a selfish manner and system-oriented which are more platform dependent. The former perform as much as possible user-space calls, decreasing its dependence with underneath platform while the latter triggers and relocates functionality to kernel.

B. L2 Transmission Queue

Before presenting our last evaluation metric, it is convenient to preliminarily present how Linux handles outgoing frames at layer 2.

The main function of the kernel at the link layer is scheduling the packet to be sent out. For this purpose, Linux uses the queueing discipline (struct Qdisc) abstraction. The dev_queue_xmit method puts the sk_buff on the device queue string using qdisc-¿enqueue virtual method. If it is necessary (when the device doesn’t support scattered data) the data is linearised into the sk_buff but this requires copying the data to different memory buffer. Several Qdisc scheduling policies exist. The basic and most used on is pfifo_fast, which has three priorities granularity. The device output queue is immediately triggered with qdisc_run() method and it calls qdisc_restart(), which takes as sk_buff from the queue using qdisc-¿dequeue virtual method. Specific queueing disciplines may delay sending by not returning any sk_buff and setting up a qdisc_watchdog_timer() instead.

TABLE IV. 1ST ORDER BURST METRICS.

<table>
<thead>
<tr>
<th>metric</th>
<th>iperf</th>
<th>mgen</th>
<th>c/rude</th>
<th>D-ITG</th>
</tr>
</thead>
<tbody>
<tr>
<td>100B, 90%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length (ms)</td>
<td>2.4178</td>
<td>0.1026</td>
<td>7.026</td>
<td>0.075</td>
</tr>
<tr>
<td>Size (KB)</td>
<td>26.8836</td>
<td>1.11929</td>
<td>80.9527</td>
<td>0.78109</td>
</tr>
<tr>
<td>100B, 50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length (ms)</td>
<td>0.8653</td>
<td>0.1554</td>
<td>5.574</td>
<td>8.764</td>
</tr>
<tr>
<td>Size (KB)</td>
<td>0.78109</td>
<td>0.571852</td>
<td>34.3898</td>
<td>1.47691</td>
</tr>
<tr>
<td>100B, 25%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length (ms)</td>
<td>0.3743</td>
<td>46.3683</td>
<td>0.3464</td>
<td>45.484</td>
</tr>
<tr>
<td>Size (KB)</td>
<td>0.81594</td>
<td>1.96481</td>
<td>0.837289</td>
<td>2.07256</td>
</tr>
<tr>
<td>1470B, 90%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length (ms)</td>
<td>1.1727</td>
<td>1.4201</td>
<td>1.2706</td>
<td>2.0391</td>
</tr>
<tr>
<td>Size (KB)</td>
<td>9.97668</td>
<td>7.18811</td>
<td>12.9837</td>
<td>5.02086</td>
</tr>
<tr>
<td>1470B, 50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length (ms)</td>
<td>1.8307</td>
<td>6.1344</td>
<td>2.7212</td>
<td>5.6885</td>
</tr>
<tr>
<td>Size (KB)</td>
<td>8.96854</td>
<td>5.96565</td>
<td>15.8702</td>
<td>5.05238</td>
</tr>
<tr>
<td>1470B, 25%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length (ms)</td>
<td>13.7732</td>
<td>4.7424</td>
<td>7.4846</td>
<td>12.7318</td>
</tr>
<tr>
<td>Size (KB)</td>
<td>4.50435</td>
<td>8.68084</td>
<td>22.1778</td>
<td>4.67261</td>
</tr>
</tbody>
</table>

Depends on particular network devices capabilities. Small buffer sizes offer in general low delay but are inadequate to absorb any packet train occurrence in the network increasing loss probability as well. The table IV summarizes the results of the default configuration of burst detection algorithm. We present the average length of burst expressed in time (ms), the size in bytes during the experiment. The results are averaged out total 5 repetitions of same use case scenario.
When the timer expires, netif_schedule() is called to start transmission. Eventually, the sk_buff is sent with dev_hard_start_xmit() and removed from the Qdisc. If sending fails, the skb is re-queued. netif_schedule() is called to schedule a retry. The later method raises a software interrupt, which causes net_tx_action() to be called when the NET_TX_SOFTIRQ is ran by ksoftirq. net_tx_action() calls qdisc_run() for each device with an active queue. Since all traffic generators considered in this work are user space applications, we argue that when a packet reaches at the outgoing transmission queue inside the kernel, the generator is not involving at this part of creation and packet transmission process. As described above, the system is responsible to handle the interrupts and hardware needs for actual frame departure. Though, using tc (Traffic Control) [10] tool we have observed that each TG handles differently the outgoing buffer. MGEN is more user space oriented software. It is not rely on underneath system and need to have more control of generation process. On the other hand, IPERF, RUDE and D-ITG are depend on system and generate bursty traffic inside the kernel, shifting responsibility towards to operating system to meet timing restrictions.

**REFERENCES**


**VII. CONCLUSIONS**

As underlined in the introduction, bursting activity plays an important role during to any under development MAC protocol, and it is greatly influenced by network traffic generator implementation and design. A first step in the characterization of the network behaviour is the capability to reliably detect and characterize such an activity. The choice of these tools is motivated by the fact that they produce very specific burst patterns, as shown by the spike trains represented above.

In this work, we analysed the behaviour of widely used by research communities network traffic generators. We have shown that for particular experimental scenario and different network tuning, one NTG could outperform the others in terms of flow reliability and determinism by generating more accurately the traffic profile that has been defined by the user.