ACCEPTED FROM OPEN CALL

Do You Trust Your Software-Based Traffic Generator?

Alessio Botta, Alberto Dainotti, and Antonio Pescapé, University of Napoli Federico II

ABSTRACT

Networking research often relies on synthetic traffic generation in its experimental activities; from generation of realistic workload to active measurements. Often researchers adopt software-based generators because of their flexibility. However, despite the increasing number of features (e.g., replication of complex traffic models), they are still suffering problems that can undermine the correctness of experiments: what is generated is sometimes far from what is requested by the operator. In this article, by analyzing four of the most used packet-level traffic generators in literature, we show how they fail to follow the requested profiles. Moreover, we identify and discuss key concepts affecting their accuracy as well as mechanisms commonly adopted to improve it. This contribution goes toward improving the knowledge researchers and practitioners should have of the tools used in experimental works, and at the same time illustrates some directions for the use and design of software-based traffic generators.

INTRODUCTION

Research on computer networks has always relied on synthetic traffic generation, that is, *injection of packets into a network in a controlled fashion*. This is usually done to perform a measurement experiment replicating traffic generated by common network applications or by the control plane: examples are the generation of average packet rates (or bit rates) for the analysis of devices under test, or the reproduction of background traffic to experiment with network protocols under different load conditions. Moreover, the validity of statistical approaches for the study of computer networks has brought about significant work in several fields that often intersect, making crucial the ability to perform experiments with synthetic traffic replicating precise statistical properties [1–4].

Traffic generators are implemented over both hardware and software platforms. Hardware platforms are typically more precise and expensive, reach better performance, and are generally developed by firms;[1] software platforms are less precise, cheaper, and generally developed by research units or universities. Moreover, software traffic generators are often open source and freeware. Most of the time, researchers in the networking field use software platforms, not only for economical reasons, but mainly for their flexibility:

- Easy deployability of multiple nodes (even hundreds) to reproduce distributed scenarios
- Ability to rapidly modify and extend the code for a specific research purpose, adding new features, statistical models, and support of new operating systems[2] and hardware platforms[3]
- Possibility to perform more realistic experiments and test actual implementations by running on top of real operating systems and network protocol stacks

On the other hand, while for hardware traffic generators it is common practice to have detailed datasheets containing certified information such as confidence intervals of the imposed values (e.g., bit rate), software platforms cannot provide the same information. This is because their metrological properties (i.e., *accuracy* of the traffic generation process) depend on the commercial off-the-shelf (COTS) hardware used, the operating system adopted, and the status of the host used for traffic generation. Therefore, in a measurement experiment lacking a preliminary analysis of such metrological properties, the reference (i.e., injected input) remains uncertain, and consequently results could be invalidated. This statement has opened and stimulated fresh discussions during recent flagship events in the networking community. The problem, indeed, is that studies of the networking research community often rely on the use of software generation platforms, and they take for granted the imposed *traffic profile*. Understanding their accuracy is therefore a fundamental aspect to be considered [5], as they are actually used as measurement and experimentation instruments. This work points the attention on *if and how much* these tools can be reliable in generating the user-requested traffic profiles, a problem underestimated in literature.

TRAFFIC GENERATION AND NETWORKING RESEARCH

Approaches for the quantitative evaluation of the performance of computer networks can be classified into three categories: *analytic*, *simulative*, and *experimental*. In spite of the great inter-

[1] *Agilent HP, Candelatech, IXIA, Skaion, Omnicor, Spirinet, ZTI-Telecom, …*

[2] *Linux Familiar, Snapgear, Montavista, UCLinux, Openwrt, …*

[3] *ARM, XScale, Cell, Cavium, …*

0163-6804/10/$25.00 © 2010 IEEE

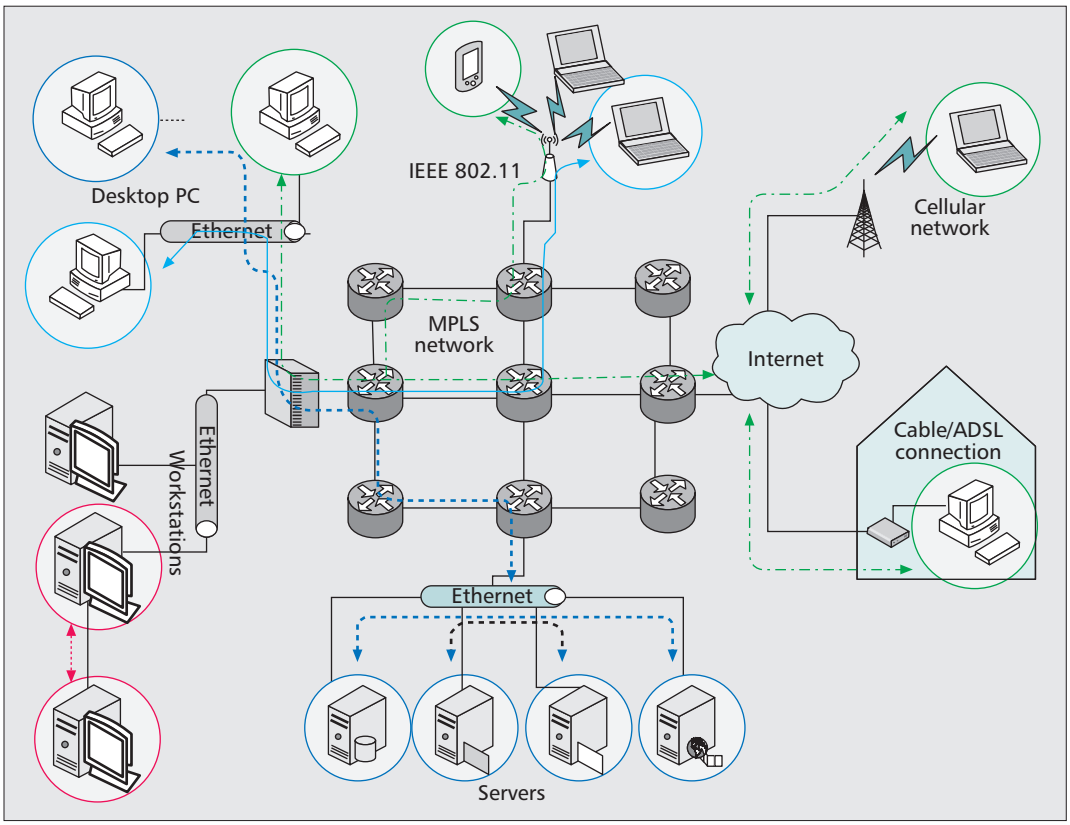

Figure 1. *Possible scenarios used for experimentations.*

We focus on the accuracy of generation defined as the measure of the difference between the requested traffic profiles (i.e., imposed rates and statistical distributions) and those actually generated, emphasizing the need for more rigor when choosing and using such tools.

est in the first two approaches, in the last years passive and active experimental measurements have been largely used in both locally[4] and geographically[5] distributed testbeds. As for the active ones, we notice that the attention is usually paid only to the output [6], that is, the results provided by the active tool, whereas few or no considerations at all are made on the *quality* of the input. It is the authors' opinion that this is probably a consequence of active measurement software not being recognized as *measurement instruments*. However, if such input is uncertain some of the foundations of the experimental approach fail:

- The repeatability of the same experiment may be impracticable.
- A comparison between two different experiments may be infeasible.
- The results may be related to wrong assumptions.

In traffic generation the statistical profiles to be followed are carefully chosen according to models derived from theoretical or experimental studies [7], but no attention is usually paid to what is actually injected into the network. In this work we focus on the accuracy of generation defined as the measure of the difference between the requested traffic profiles (i.e., *imposed rates* and *statistical distributions*) and those actually generated, emphasizing the need for more rigor when choosing and using such tools. We consider four of the software-based traffic generators most widely used by the scientific community, and we show some of their current limitations, in our opinion too often underestimated by researchers. We identify some key concepts affecting accuracy and, more generally, undermining the basis of the experimental method. In addition, we illustrate common counter measures to adhere to the imposed traffic profiles.

The networking research community is now mature enough to consider this issue, particularly relevant in experimentations such as (see Fig. 1 for a reference scenario):

- Comparison of different transport protocols carrying realistic traffic loads and under realistic background traffic
- Testing of quality of service and routing architectures
- Active measurements of link delay (and other parameters) using Poisson probing
- Queue performance analysis of intermediate nodes when serving multiple flows of realistic traffic
- Estimation of available bandwidth
- Testing of statistical-based anomaly and intrusion detection systems

The large amount of work using software-based traffic generators testifies that they are heavily adopted in the networking research field: querying the IEEE search engine[6] for published papers matching the string traffic generator reported an average of 30 per year until 2001 and jumps to about 150 since 2002, with a peak of about 200 in 2005. In addition, a simple search on both Google and Google Scholar of works using only the four traffic generators used in this work results in a very large number of papers, tutorials, and technical reports (see the last column of Table 1 for the actual numbers as of July 2008).

[4] *Orbit Testbed: http://www.orbit-lab.org/; DETERlab Testbed: http://www.isi.edu/deter/*

[5] *Planetlab: http://www.planet-lab.org/; OneLab: http://www.one-lab.org/*

[6] *http://ieeexplore.ieee.org*

Traffic generators	Operating systems	Network protocols	Transport protocols	Available distributions	Measured parameters	Hits on Google and Google Scholar
MGEN	Linux, FreeBSD, NetBSD, Solaris SunOS, SGI, DEC	IPv4	UDP, TCP	Constant, exponential, on/off	Throughput, packet loss, delay, jitter	473
TG	Linux, FreeBSD, Solaris SunOS	IPv4	UDP, TCP	Constant, uniform, exponential, on/off	Throughput, packet loss, delay, jitter	125
RUDE/ CRUDE	Linux, Solaris SunOS, and FreeBSD	IPv4	UDP	Constant	Throughput, packet loss, delay, jitter	286
D-ITG	Linux, Windows, Linux Familiar, Montavista, Snapgear	IPv4, IPv6	UDP, TCP, DCCP, SCTP, ICMP	Constant, uniform, exponential, pareto, cauchy, normal, poisson, gamma, on/off	Throughput, packet loss, delay, jitter	300

Table 1. *Traffic generators considered in the experimental evaluation.*

PLAYING WITH TRAFFIC GENERATORS

While a comprehensive list of traffic generators is reported on the ICIR website,[7] the following list represents a smart taxonomy underlining both their architectural features and the abstraction layer at which they work:

Application-level traffic generators: They emulate the behavior of specific network applications in terms of the traffic they produce. Surge[8] (capable of generating traffic by emulating the interactions between hundreds of web clients and servers) is an example of this category.

Flow-level traffic generators: They are used when the replication of realistic traffic is requested only at the flow level (e.g., number of packets and bytes transferred, flow duration). For example, Harpoon[9] uses a set of statistical parameters that can be automatically extracted from Cisco Netflow traces.

Packet-level traffic generators: With this term we refer to generators based on packets Inter departure time (IDT) and packet size (PS). The size of each packet sent, as well as the time elapsed between subsequent packets, are chosen by the user, typically by setting a statistical distribution for both variables. A lot of largely used traffic generators work at the packet level,[10] and they can also be used as active measurement tools (e.g., the delay and jitter experienced by each packet can be recorded). In this category also fall generators that do not operate with standard sockets but typically forge packets starting from layer 2 to upper layers and generators used as bandwidth measurement tools. Examples of the first category are Brute[11] and KUTE,[12] working at the kernel level, or PktGen[13] and IXPKTGEN[14] specifically developed for network processor architectures. Examples of the second category are Iperf[15] andNetperf[16]: such tools usually work by sending as much traffic as possible to measure network performance, but they are not strictly considered traffic generators because they cannot generate specific traffic profiles requested by the operator.

Closed-loop and multilevel traffic generators: Several frameworks (e.g., Swing)[17] have been proposed to take into account the interactions among multiple layers of the protocol stack (users, sessions, connections, and network characteristics). Research works regarding their design have been presented, whereas these kinds of software are usually not made available to the scientific community, or (due to their complexity) are rarely used to conduct experimental researchs.

In this work we have chosen some of the most widely used packet-level traffic generators, without considering both generators forging packets at layer 2 and bandwidth measurement tools: we are interested in traffic generators that inject synthetic traffic in a realistic fashion and use the protocol stack of the operating system on which they run as a common network application would do.

Table 1 contains the main features and additional information on the four considered platforms: *TG*[18] from ISI, *MGEN*[19] from the U.S. Naval Research Laboratory, *RUDE/CRUDE*[20] from Tampere University of Technology, and *D-ITG*[21] from the University of Napoli. For our tests we used two Pentium IV machines, equipped with Linux 2.6.15 and connected back-to-back with a cross cable to their PCI Gigabit Ethernet interfaces (see the dotted red line in Fig. 1). We switched off all the unnecessary applications, such as X-Windows. Obviously, many other OS-level processes were still competing for resources. With this simple testbed, we analyzed the ability of the generators to replicate the traffic profiles requested by the operator in terms of both imposed bit rate (and packet rate) and statistical distributions of packet IDT. We found that they generate traffic profiles that are very different from what is supposed to be injected into the network. To obtain the results reported in Figs. 2–5, we analyzed the log files produced by each traffic generator related to the packets sent: the values reported in the following represent an upper bound for what is observable on the network or at the receiving side. From such log files we extracted the sequence number, size, and sending time for each generated pack-

[7] *http://www.icir.org/models/traffic generators.html*

[8] *http://cs-www.bu.edu/faculty/crovella/links.html*

[9] *http://pages.cs.wisc.edu/~jsommers/harpoon/*

[10] *TG, MGEN, RUDE/CRUDE, D-ITG, UDPgen, Traffic, PacGen, NTGen, UDPGenerator, IPGen, Poisson Traffic Generator, MxTraf*

[11] *http://netgroup-serv.iet.unipi.it/brute/*

[12] *http://caia.swin.edu.au/genius/tools/kute*

[13] *http://www.tnt.dist.unige.it/np/index.php/PktGen*

[14] *http://protocols.netlab.uky.edu/~esp/pktgen/*

[15] *http://dast.nlanr.net/Projects/Iperf/*

[16] *http://www.netperf.org/netperf/*

[17] *http://www.cs.ucsd.edu/~kvishwanath/Swing/*

[18] *http://www.postel.org/tg/tg.htm*

[19] *http://cs.itd.nrl.navy.mil/work/mgen/index.php*

[20] *http://rude.sourceforge.net/*

[21] *http://www.grid.unina.it/software/ITG*

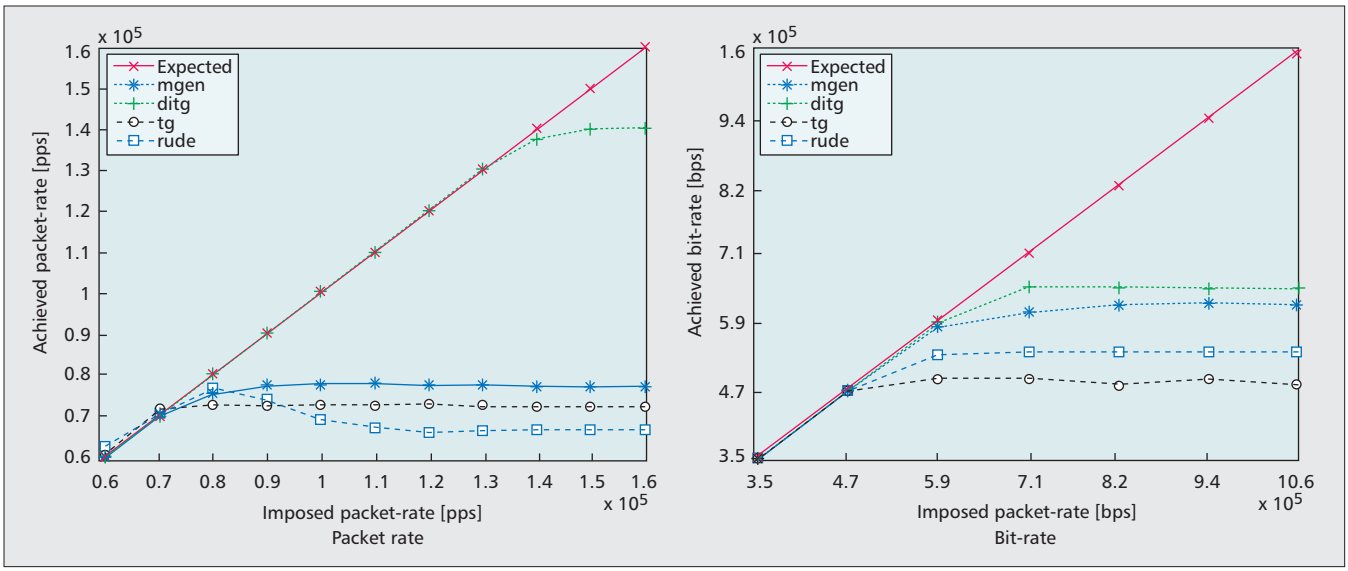

Figure 2. *Achievable packet-and bit-rate.*

et. This allows us to evaluate the number of packets and bytes generated as well as the packet IDT.

Figure 2 reports bit and packet rates achieved by the four considered traffic generators (y axis) sketched against the expected behavior (x axis). The points reported in the diagrams represent the average values over experiments 2 min long. To evaluate the tools working in their best conditions, we used the minimum PS allowed by each generator (from 16 to 20 bytes) for the packet rate, and the maximum PS (i.e., 1472 bytes of UDP payload) for the bit rate. As regards the packet rate (right plot of Fig. 2) when asked to generate from 60 to 160 kpackets/s, starting from a certain point, all the generators significantly deviate from the expected behavior. More precisely, we can observe that the generators behave differently. TG and MGEN after reaching their maximum values, at about 70 kpackets/s, saturate to a fixed packet rate as the requested rate increases. RUDE/CRUDE after reaching the maximum value, presents a decreasing packet rate. D-ITG starts to deviate from the expected behavior at about 130 kpackets/s. These differences testify that beyond hardware constraints, software design has a significant impact on the ability of the generators to match the requested packet rate.

As for the bit rate, the right plot of Fig. 2 shows the results we obtained when the tools were asked to generate from 350 Mb/s to 1 Gb/s. Also in this case, we observe a departure from the requested rate starting from a certain point (500 Mb/s), which looks like a saturation of the throughput capabilities of all the generators, and a different behavior of the considered generators. It is interesting to note that, working with full packets, this saturation point corresponds to a packets per second value much lower than that achieved in the packet rate tests, which were made with minimum-sized packets (e.g., 500 Mb/s correspond to 44 kpackets/s). This highlights a difference between software-based traffic generators and architectures with application-specific hardware, which treat packets as single units, using dedicated buffers. Filling packets and copying their content is an expensive operation in software-based generators; thus, strategies that contain CPU usage can significantly relieve the system and improve accuracy.

Both analyses testify that, in experiments requiring the injection of a specific throughput (in terms of packets or bytes), we cannot take for granted a value that could be, in certain cases, different from the one imposed, and therefore that ill-behaved generators can affect the results of an experiment.

In experiments requiring the replication of realistic traffic profiles (e.g., voice or video traffic), the generation of traffic with specific statistical distributions of IDT and PS is crucial. While for PS it is obvious that the actually generated values are identical to those imposed, for IDT this is not true because several factors can affect correct timing when sending packets. The resulting traffic profile may therefore be very different from what was needed by the experiment. This also applies to active measurements: recently, in the research community the subject of exponentially-distributed packet probing in active measurements has been given new attention. It is widely accepted indeed, that a methodology using Poisson processes should guarantee unbiased estimation, for example when measuring delay on network paths. In [8] it has been argued that more important than unbiased estimation is the minimization of the mean-square error (MSE), which depends on both the estimation bias and variance. And, in terms of MSE, Poisson probing is not necessarily optimal. This has generated a lot of discussion and new research activities. Despite its relevance, this discussion neglects that actual tools used by researchers and practitioners may not reliably replicate the requested statistical properties. Indeed, as we show in the rest of this section, even if a specific distribution is chosen, the generation process can be poisoned.

Figure 3 reports some distributions of IDT produced by the considered generators compared against the imposed ones. Such figures allow to be verifed the real values of IDT generated by the tools. The top plot reports the results obtained when generating 8 Mb/s of constant bit rate traffic by using about 1950 packets/s with 512 bytes of UDP payload. Even at such a low rate, TG and MGEN with polling mode disabled[22] (`mgen p-off` in the figure) are not able to strictly follow the imposed IDT. Indeed, for both tools the IDT attain values that are either very small (i.e., < 10 μs) or around 4 ms. The middle plot shows exponentially distributed IDT with 1 Mb/s average bit rate. Again, MGEN without polling and TG are not able to follow the analytical distribution, and present spikes all over the reported range. The bottom plot reports the same distribution but with 8 Mb/s of average rate (i.e., about 1950 packets/s). In this case the output of MGEN without polling and TG resembles a bimodal distribution, which is very different from the expected one.

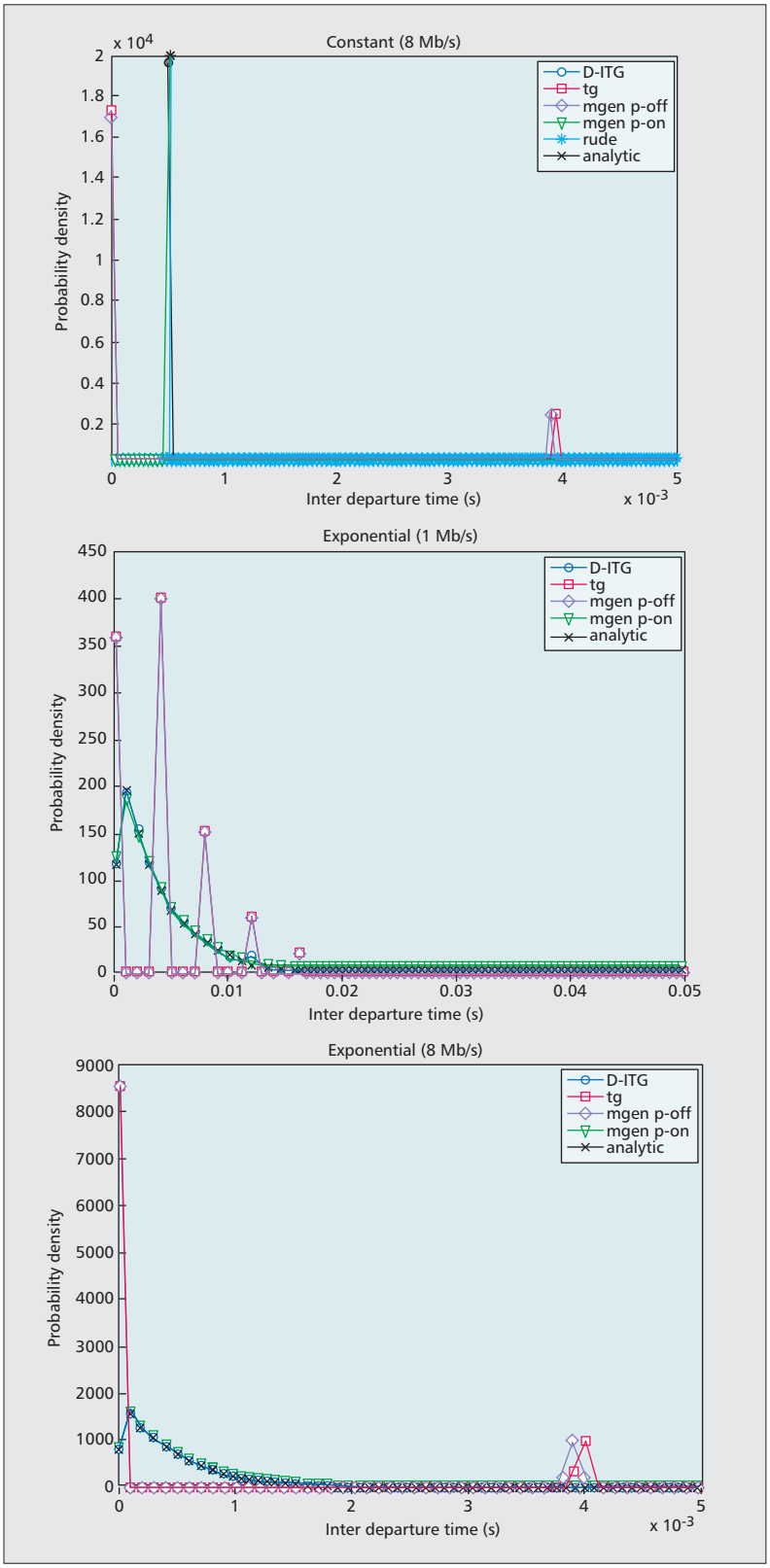

Figure 3. Accuracy *of generated PDFs (at low rates).*

It is worth noting that the packet and bit rates here considered are much lower than those for which the generators started to deviate from the requested behavior in the achievable throughput tests. This means that the IDT generation process is often poisoned even starting from such low rates (i.e., within the stable working range of the generator). Indeed, as the requested rate becomes higher all four of the generators increasingly deviate from the expected distribution. This phenomenon is quantitatively observable in Fig. 4, where a discrepancy measure (λ^2) [9] between the obtained empirical distributions and the imposed one is reported for constant, exponential, and uniform distributions.

We conclude that, also for IDT distributions (and consequently for the traffic profiles), we cannot take for granted distributions that could be very different from those imposed; then again, ill-behaved generators can affect the results of an experiment.

Traffic Generators: Looking into the Box

A closer look at the traffic generators considered, when running in practice, shows a complex scenario with several tasks competing for hardware resources and managed by the operating system (OS). Aside from the main task a box generating synthetic traffic should ideally do — injecting packets into the network on time — other concurrent tasks are sources of what we define as external and internal interference. By external we mean interference between generation and other processes on the same machine; by internal we mean interferences among different tasks carried out by the traffic generator (time-stamping, logging, calculations, etc.). The impact of both types of interference is dual and it results in inaccuracies in the packet/byte rate generated and the distributions of IDT such as those highlighted in the previous section. First, they consume resources, reducing the availability of CPU time and buses. Second, they interfere, depending on the choices made by the OS scheduler, with the correct timing of sending packets. This can be observed in an imprecision of the

[22] *The option is called* `precise off` *in the tool documentation.*

`select()` (or `sleep()`) calls that are used by the generators to set the IDT timers.

In order to look in more detail at such phenomena, we added probes to the code of the generators that verify the result of each `select()` call, examined the number of context switches, and studied several mechanisms that are implemented in the generators for improving accuracy.

External interference typically delays the return from a `select()` call when a context switch happens. In the case of D-ITG generating 590 Mb/s throughput (with logging disabled), we observed that the number of times the `select()` call returned in late (20 μs was set, whereas around 40 μs elapsed before the call returned) is proportional to the number of context switches (every 90 ms). Running a graphical user interface waiting for user login doubled the entity of such delays without changing the number of their occurrences or the number of context switches. The imprecision of the `select()` timeout due to context switches is also visible at much lower rates. In Fig. 3 we consider rates from 1 to 8 Mb/s and focus on the statistical distributions of IDT instead of throughput: the IDT distributions obtained by MGEN p-off and TG show several imprecise `select()` timeouts at multiples of about 4 ms. This behavior can be explained with the scheduling operated by the OS under such relatively low load conditions. The process is typically descheduled from the CPU when the `select()` is called and the control is returned to the process within a certain amount of time, also depending on the slices the scheduler is assigning to processes under the current load conditions. To overcome this problem, both D-ITG and MGEN (with the option `p-on` enabled) implement a polling mechanism, for timeouts smaller than 10 ms, which cycles until the expiration of the timeout instead of calling a `select()`. This is to force the process to stay on the CPU. The results are visible in Figs. 3 and 4: the obtained IDT distributions become very close to the analytical, and remain accurate for different rates.

As for *internal interference*, in Fig. 2 we observe that all traffic generators reach a different throughput boundary. While there are obviously hardware constraints, the practical limit of the boxes used in our experiments is much higher than the results achieved by all generators: by piping data from memory into the netcat utility[23] we achieved an average throughput of about 900 Mb/s. This shows that, aside from external interference and hardware constraints, there are limitations of the software generators due both to their software design and the tasks they must complete. Such traffic generators indeed have to perform several operations. First, they usually set each IDT as a sample of a statistical distribution and, even when the IDTs are constant, the generator is in charge of verifying the amount of time elapsed before sending each new packet. Moreover, the generators typically fill packet contents with useful information, such as IDs and timestamps, instead of simply copying data from memory. Finally, because such tools are used for conducting experiments, they need to log information regarding the sent and received packets. The impact of logging data to disk is particularly high: by enabling logging in D-ITG generating a 590 Mb/s throughput (Fig. 2) we observe a dramatic increase of delayed returns from the `select()` call (about 1000/s instead of 15), whereas the number of context switches does not increase notably — this was expected because the interference is internal. Traffic generators try to reduce the impact of logging by enabling optimized mechanisms. D-ITG temporarily stores the logging information related to

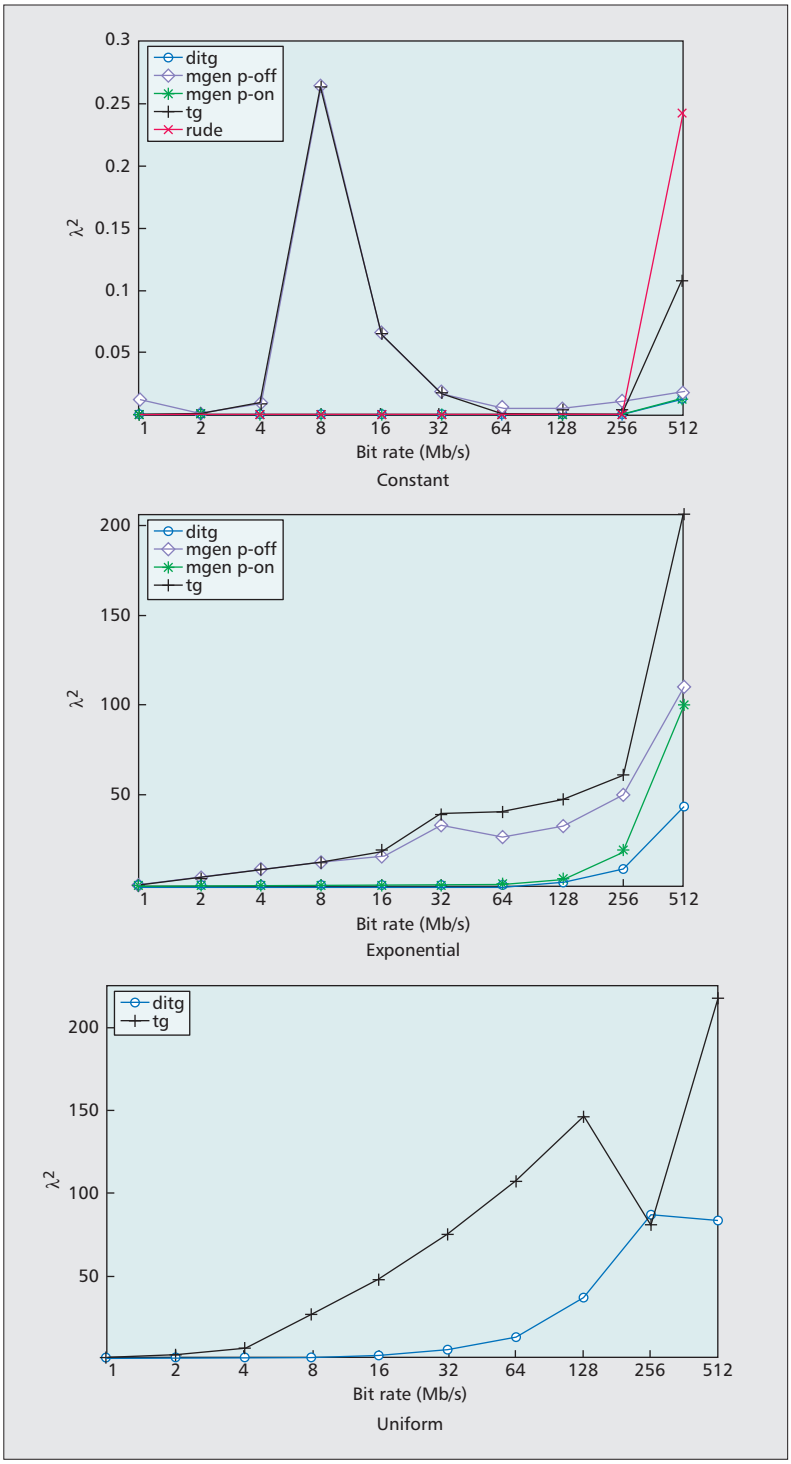

Figure 4. Accuracy *of the IDT measured with the λ² (at different rates).*

[23] *The command* nc –c 'cat /dev/mem' –u 192.168.1.2 9 *copies 4-kBytes blocks from memory and sends them as UDP payload to the* discard *service of the receiver box.*

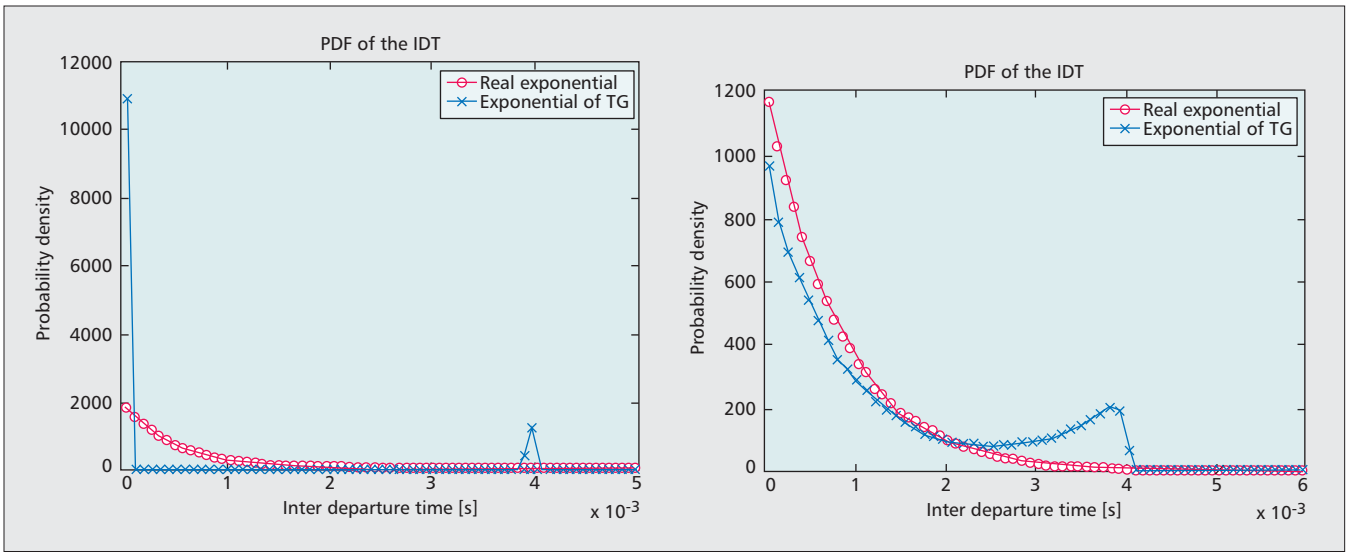

Figure 5. *Jitter measured with probing traffic having accurate and inaccurate exponential IDT.*

a set of packets in a memory buffer, which is dumped on disk when full. Moreover, to reduce the quantity of data to be dumped and the number of operations to be made, both D-ITG and TG dump data directly in binary format (instead of text format as traffic generators sometimes do), which must then be decoded at the end of the experiment.

To cope with the delayed returns from the `select()` and in order to achieve a higher throughput, some of the considered generators (e.g., D-ITG, MGEN, and TG) implement a mechanism we call *IDT recovery*: the amount of delay is checked each time the `select()` returns; if the delay is so high that new packets should have been sent during that time in order to satisfy the requested throughput, such packets are immediately sent without any timeout. This way, the actual average packet rate gets closer to the requested one. For the example reported earlier — D-ITG with logging enabled and 590 Mb/s average throughput — we verified that the IDT recovery mechanism is used almost each time a delayed return from the `select()` happens, with an average of two packets recovered each time. When the imposed rate is instead of 710 Mb/s (the first point of the saturation limit in Fig. 2), we observe that the generator is constantly in IDT recovery mode, meaning that the generation cycle (which includes all the tasks the generator must carry out) is never able to send packets in time (about 16 μs per IDT).

The analysis here presented has shown that there are several sources of interference affecting accuracy, which are not easy to track down and not completely controllable. Moreover, their existence and characteristics depend on the operating conditions (e.g., traffic load generated) as well as the hardware and OS configuration. Some traffic generators implement mechanisms to improve accuracy, which partially counter some of them. The understanding of such issues and partial countermeasures can help the researcher in properly conducting an experiment involving synthetic traffic generation.

Discussion

To overcome the limitations of simulative and analytical studies [10], experimental approaches are adopted in several contexts of networking research. In such approaches the need for software-based traffic generators, capable generating *synthetic* — but *realistic* — network traffic, is often indisputable. The analysis here presented has shown how the accuracy of these tools is sometimes far from what is expected. This can heavily affect experimental results without operator awareness. As an example, let us consider a simple situation: the measurement of the jitter over a network link. The evaluation of jitter is particularly important in studies related to the quality of service (QoS) of network architectures. More precisely, it represents a crucial parameter in the case of multimedia communications. Inaccurate jitter measurements can bring about:

- Incorrect quantitative performance evaluation of QoS parameters
- Erroneous channel models
- Wrong assumptions on the user-perceived QoS

As probing traffic we use a traffic profile with exponential IDT and PS (512-byte average), and we observe the difference between the ideal case and a real (but inappropriate) one by simulating both the traffic generation process and the network under test with *ns*2. In the top plot of Fig. 5 we show the two IDT distributions we used: an exponential distribution, and the one obtained from the logs of TG. The average bit rate is equal to 8 Mb/s in both cases. The bottom plot of Fig. 5 shows the probability density function (PDF) of the jitter experienced by the packets in the two situations when traversing a single duplex link with drop tail behavior and implementing an M/M/1 queue. Even in such a straightforward configuration as this (a single link, a single node generating traffic, etc.), it is clear that the impact of inaccurate traffic profiles can significantly alter the results of the measurement. The two PDFs of the jitter are quite different. In addition, other

statistics are different: the average is equal to 8.1*e* – 04*s* in the ideal case and equal to 13.0*e* – 04*s* in the case of TG, whereas the autocorrelation is respectively equal to 0.0052 and 0.0464, respectively.

Aside from the presence of relevant deviations from the requested traffic profiles, we have also shown that traffic generators implement mechanisms that allow accuracy to be improved under some circumstances. Operators should be aware not only of their existence, but also of their relation to the problems they are trying to counter and thus their potential impact on the experiment. Indeed, their improper use may even cause a further degradation of accuracy. An example is the IDT recovery mechanism explained in the previous section: while such mechanism is effective in reaching an average throughput closer to the requested one, it may negatively impact accuracy of the distribution of the IDT. It is evident, indeed, that the IDT recovery makes the traffic generator ignore the IDT values extracted from the analytical distribution instead causing bursts of packets. This is why such a feature can be optionally enabled in some generators (e.g., D-ITG).

Conclusion

We think the investigation provided in this article highlights issues often underestimated by researchers and practitioners, and can help them increase their attention in experimental studies comprising traffic generators and in general using active measurement approaches. We have pointed out the lack of accuracy of current traffic generators and that, despite the adopted countermeasures, generated traffic profiles can still be very different from those requested. The analysis here provided is useful to:

- Understand some hidden mechanisms of traffic generation
- Design and implement more precise traffic generation platforms
- Improve the awareness of users and give guidelines to properly choose and use software-based traffic generators

Therefore, a researcher can either improve accuracy by properly choosing and configuring the right tool, or assess if an incorrect or uncertain input invalidates the specific experimental setup.

References

[1] F. Dressler, "Policy-Based Traffic Generation for IP-Based Networks," *IEEE INFOCOM*, Apr. 2006.

[2] K. V. Vishwanath and A. Vahdat, "Realistic and Responsive Network Traffic Generation," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 36, no. 4, Oct. 2006, pp. 111–22.

[3] A. Botta, A. Dainotti, and A. Pescapé, "Multi-Protocol and Multi-Platform Traffic Generation and Measurement," *IEEE INFOCOM*, Demo session, May 2007.

[4] L. Ciavattone, A. Morton, and G. Ramachandran, "Standardized Active Measurements on a Tier 1 IP Backbone," *IEEE Commun. Mag.*, vol. 41, no.6, June 2003, pp. 90–97.

[5] M. Paredes-Farrera, M. Fleury, and M. Ghanbari, "Precision and Accuracy of Network Traffic Generators for Packet-by-Packet Traffic Analysis," *2nd Int'l. TridentCom*, Barcelona, Spain, Mar. 2006.

[6] F. Michaut and F. Lepage "Application-Oriented Network Metrology: Metrics and Active Measurement Tools," *IEEE Commun. Surveys & Tutorials*, vol. 7, no. 2, Apr. 2005.

[7] C. Williamson, "Internet Traffic Measurement," *IEEE Internet Comp.*, vol. 5, no.6, Nov./Dec. 2001, pp. 70–74.

[8] F. Baccelli *et al.*, "The Role of PASTA in Network Measurement," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 36, no. 4, Oct. 2006.

[9] S. Pederson and M. Johnson, "Estimating Model Discrepancy," *Technometrics*, vol. 32, no. 3, Aug. 1990, pp. 305–14.

[10] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Trans. Net.*, vol. 9, no. 4, Feb. 2001, pp. 392–403.

Biographies

Alessio Botta [M] received his M.S. Laurea degree in telecommunications engineering in 2004 from the University of Napoli Federico II, Italy and his Ph.D. in computer engineering and systems from the same university. From February to August 2009 he visited the Networking and Security Department at Eurécom, Sophia Antipolis, France, working on the monitoring of 3G networks. Currently he holds a post-doctoral position at the Department of Computer Engineering and Systems of the University of Napoli Federico II. His research interests fall in the area of networking, with specific regard to network monitoring and measurements.

Alberto Dainotti [M] received his M.S. Laurea degree in computer engineering in 2004 from the University of Napoli Federico II. In 2008 he received his Ph.D. in computer engineering and systems from the same university. From July 2007 to February 2008 he visited the Cooperative Association for Internet Data Analysis (CAIDA) at the University of California, San Diego, working on traffic classification and analysis of malware traffic. Currently he works as a post-doctoral at the Department of Computer Engineering and Systems of the University of Napoli Federico II. His research interests fall in the areas of network measurements, traffic analysis, and network security.

Antonio Pescapé [SM] (pescape@unina.it) is an Assistant Professor at the Department of Computer Engineering and Systems of the University of Napoli Federico II. He received his M.S. Laurea degree in computer engineering and his Ph.D. in computer engineering and systems, both from the University of Napoli Federico II. His research interests are in the networking field with focus on models and algorithms for Internet traffic, network measurements and management of heterogeneous IP networks, and network security. He has coauthored over 80 journal and conference publications. He has served and serves on several technical program committees of IEEE and ACM conferences (IEEE GLOBECOM, IEEE ICC, IEEE WCNC, IEEE HPSR, etc.). He also serves as an Editorial Board Member of *IEEE Communications Survey & Tutorials* and was Guest Editor for *Computer Networks* (Special Issue on Traffic Classification and Its Applications to Modern Networks).

We have pointed out the lack of accuracy of current traffic generators and that, despite the adopted countermeasures, generated traffic profiles can still be very different from those requested.