On the use of Emulab testbeds for scientifically rigorous experiments

Christos Siaterlis, Andres Perez Garcia[1], Béla Genge
Institute for the Protection and Security of the Citizen
Joint Research Centre
Via E. Fermi 2749, 21027 Ispra (VA) Italy
e-mail: [christos.siaterlis,andres.perez-garcia,bela.genge]@jrc.ec.europa.eu

***Abstract*—Internet is considered a Critical Infrastructure (CI) that is vital for both the economy and the society. Disruptions caused by natural disasters, malicious human actions and even hardware failure pose serious risks. Emulation testbeds are increasingly used to study the Internet in order to improve protection and response mechanisms. These are frequently considered more adequate than software simulators to realistically recreate the complex behavior of networks. In this paper we study how testbeds based on the Emulab software can be used to conduct scientifically rigorous experiments, specifically in terms of: a) experiment fidelity, b) repeatability, c) measurement accuracy, and d) interference.**

Our study, which is based on extensive experimentation on different testbeds, indicates that the current trend of using emulation testbeds is justified as both realistic and efficient. We show that Emulab-based experiments are representative of real systems in terms of emerging behavior (qualitative) and that repeatable experiments are possible. The main contribution of this tutorial article is that based on experimental results we identified caveats and provided insights to significant configuration parameters and limitations that are further elaborated into a set of guidelines that any Emulab user should be aware of. Then, he/she can decide about the importance of each guideline in the context of a specific study and experiment.

***Index Terms*—Emulation, fidelity, testbeds, repeatability, network testbeds.**

I. Introduction

THE increasing dependence of Critical Infrastructures (CIs) from Information and Communication Technologies (ICT) has been recognized as a trend that might encompass significant risks to our society. Studying the resilience of such CIs, e.g., the Internet [1], and of complex cyber-physical systems [2], [3] in general is therefore an important research topic. The resilience of complex systems or system of systems could be studied by injecting faults and disruptions into real systems, software simulators or hardware emulators. Experimentation with real production systems in extreme conditions has always been difficult due to concerns about potential side-effects to mission critical services. On the other hand the development of a dedicated experimentation infrastructure with real components is often economically prohibitive. Software based simulation has always been considered an efficient approach to study physical systems, mainly because it can offer low-cost, fast and accurate analysis. Nevertheless, due to the diversity and complexity of protocols, systems and architectures of CIs, as well as the lack of specialized tools for simulating a CI, hardware-based emulation is considered a flexible and powerful approach. Emulation approaches, and specifically those based on the Emulab software [4],[5], are becoming very popular in many research areas, e.g., networking and distributed systems. Emulab is a network testbed, able to recreate a wide range of experimentation environments in which researchers can develop, debug and evaluate a complex system [6]. Emulation is particularly useful for security and resilience analysis [7], [8], because in order to study those attributes a researcher has to expose the system-under-test to high load and extreme conditions, under which, software simulators fail to capture reality.

In this paper we present a study of different characteristics of an Emulab-based testbed, namely, experiment fidelity, repeatability within and across different testbeds as well as measurement accuracy and interference. These features are necessary to conduct scientifically rigorous experiments. In order to test the fidelity we compare results from experiments employing emulation against results from experiments that use real hardware and software, i.e., without emulation. The experiments reveal how fidelity can be affected by different network load conditions and emulation specific parameters, e.g., queue size. Then, we check the repeatability of the results by comparing different experiment runs. Furthermore we study different measurement techniques in terms of accuracy and interference. To the best of our knowledge, the related work is limited and previous studies [9], [10] compare different emulation/simulation approaches rather than systematically studying the aforementioned characteristics, for example studying fidelity by comparing Emulab-based configurations against a real reference configuration. This signifies an important gap in the literature that we address in this article.

Our contribution does not only lie on the presented experimental results, but also in the transformation of our experience in terms of caveats, significant configuration parameters and limitations into a set of guidelines that researchers could use as a reference while using testbeds such as DETER [7], or Emulab in general. This would lower the barrier for new researchers trying to use Emulab and promote scientifically rigorous experimentation.

The paper is structured as an advanced tutorial and builds upon the basic concepts that were introduced by White et

1: Corresponding author. Tel:+39-0332-783720 Fax:+39-0332-789576

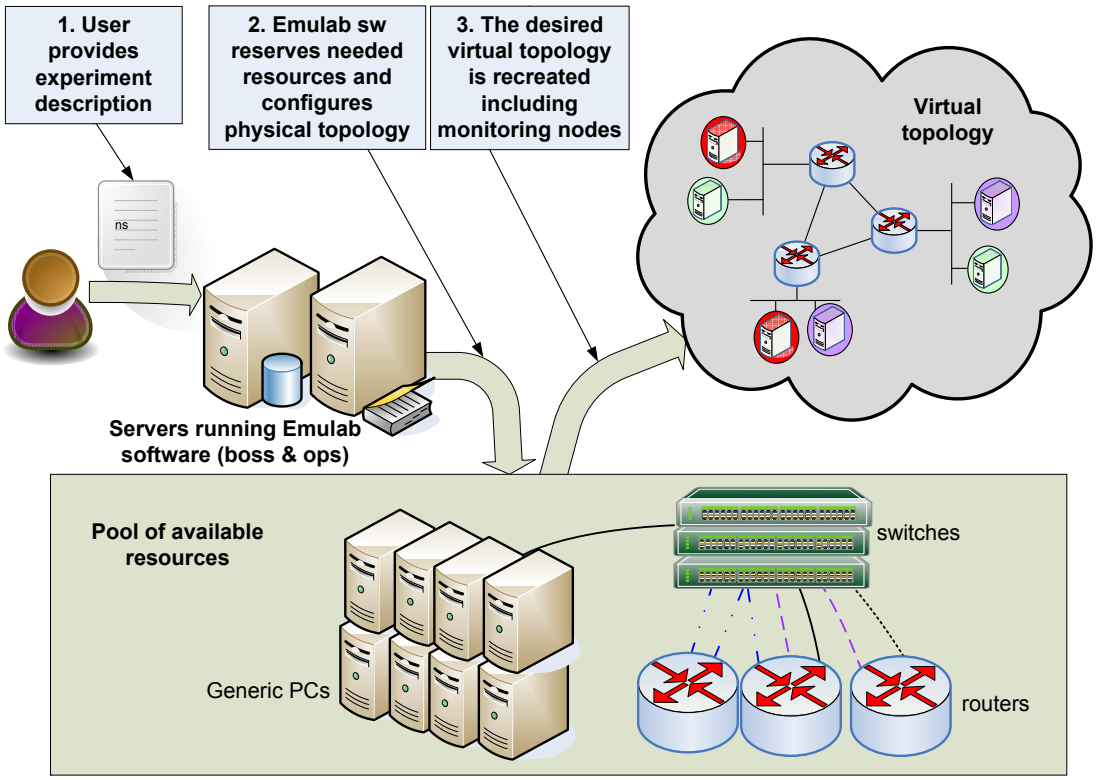


Fig. 1. Main steps for recreating a virtual network topology within an Emulab-based testbed.

al. [6], but it is not a replacement of the original tutorial provided with the Emulab software [11], [12]. We begin in Section II with a short description of a typical Emulab-based testbed and its characteristics. Then, we proceed with our study and experimental results in Section III. Here, we start with the presentation of the experimental setup and we continue addressing one by one the characteristics of fidelity, repeatability, measurement accuracy and measurement interference. We conclude in Section IV where we summarize our findings and indicate directions for future work.

II. The Emulab platform and its features

One of the most promising approaches for experimentation with large and complex systems, e.g., those found in industrial Supervisory Control and Data Acquisition (SCADA) networks [13], is the use of emulation testbeds. Pure software simulation is often too simplistic to recreate complex environments and the use of an ad-hoc testbed, where researchers have to deploy and configure manually every experimental setup, is not recommended because it is very time-consuming and error-prone to setup, maintain and change. A trend, that is constantly becoming more popular, is the use of emulation testbeds like Emulab. We have developed in our laboratory a testbed, called EPIC (Experimental Platform for ICT Contingencies), using the Emulab architecture and software, that allows us to automatically and dynamically map physical components, e.g., servers and switches, to a virtual topology. In other words, the Emulab software configures the physical topology in a way that it emulates the virtual topology as transparently as possible. This way we gain significant advantages in terms of repeatability, scalability and controllability of our experiments.

The basic Emulab architecture consists of two control servers (*boss* and *ops*), a pool of physical resources that are used as experimental nodes (generic PCs, routers or other devices) and a set of switches that interconnect the nodes. The *boss* server provides a web interface to the users and controls the testbed's internal operation, while the *ops* server provides to users a login shell from which they can access the experimental nodes and on which they can store the data required or generated by their experiments.

Once logged into the web interface, the following steps describe the experiment life cycle within our testbed (Fig. 1):

1) First we need to create a detailed description of the virtual network topology, *the experiment script*, using an extension of the NS language [14], [15]. The use of a formal language for experiment setup eases the recreation of a similar setup by any other researcher who wants to reproduce our results. In the experiment script we enumerate similar components as different instances of the same component type. This way pre-defined templates of different components (a Linux server template is a disk image with a pre-installed Linux OS) can be easily reused and automatically deployed and configured.
2) Whenever we want to run an experiment we instantiate it by using the Emulab software, which automatically reserves and allocates the physical resources that are needed from the pool of available components. This procedure is called *swap-in*, in contrast to the termination

of the experiment, which is called *swap-out*.

3) Furthermore, the software configures network switches in order to recreate the virtual topology by connecting experimental nodes using multiple VLANs. Finally, before the testbed is released for experimentation, the software configures packet capturing of predefined links for monitoring purposes.

At this point it is important to note that in step 3, the Emulab software uses two different strategies for network link emulation, e.g., delay, packet loss and bandwidth, according to the predefined instructions given in the experiment script. The *delay-node-shaping* strategy, which is the default configuration, uses extra PCs to emulate network links. These PCs, hereinafter called *delay nodes*, run Dummynet [16] to simulate link level characteristics. A different approach is taken by the *end-node-shaping* strategy that does not use extra resources and therefore runs Dummynet inside the end user's experimental nodes. In Section III-B we show that the end-node-shaping strategy can lead to unstable and unrealistic results [17].

As an example, we provide the basic experiment script that was used in our study. It recreates a 100Mbps LAN with three nodes and end-node-shaping strategy enabled. For one of the nodes we define a 10Mbps connection to the LAN in order to test different emulation speeds. For further information on the NS syntax, refer to the Emulab documentation [11], [12].

```
set ns [new Simulator]
source tb_compat.tcl

# Nodes
set node1 [$ns node]
set node2 [$ns node]
set node3 [$ns node]

# Lans
set lan0 [$ns make-lan "$node1 $node2 $node3" 100Mb 0.0ms]
tb-set-node-lan-bandwidth $node3 $lan0 10Mb
tb-set-endnodeshaping $lan0 1

$ns rtproto Static
$ns run
```

In order to conduct scientifically rigorous experiments, an Emulab testbed has to provide realistic results, appropriate measurement tools and the capacity to reproduce the experiments by a different researcher in a similar testbed. We have analyzed these properties through the following characteristics:

- **Fidelity**: refers to how accurately an experimental platform reproduces a real system, i.e., system representativeness. In many cases reproducing in an absolute way all details of a real system might not be necessary. Therefore it is preferable for an experimental platform to offer an "adjustable level of realism", meaning that one can use the level of detail that is sufficient to test the experiment hypothesis. For example, one experiment might need to reproduce a network at the very low level using real routers and real traffic (reproducing even the lower layers of the OSI model, i.e., Layer 1 and 2), while for another experiment the use of a software router and synthetic traffic generators might be sufficient (focusing for example at the application layer). The concept of adjustable level of realism gives the option to use real hardware when it is really needed, and simulators or other abstractions when not.
- **Repeatability**: represents the ability to repeat an experiment and obtain the same or statistically consistent results. Repeatable experiments require a controlled environment but to achieve them the researcher has to define clearly and in detail the initial and final state of the experiment as well as all events in between these two states. These states and events form an experiment scenario. To reproduce a previously stored experiment scenario the researcher should be able to setup the experimental platform in the initial state and trigger all necessary events in the right order and time of occurrence.
- **Measurement Accuracy and Interference**: experiments should be accurately monitored and measurements should not interfere with the experiment because they might alter the outcome of the experiment.

III. Experimental results

A. Experimental setup

In this work we study the Emulab software as a platform to conduct scientifically rigorous experiments through four characteristics, namely, fidelity, repeatability, measurement accuracy and measurement interference. Based on our results and experiences, we intend to provide assistance to future users of Emulab by developing a list of user guidelines and caveats. Ultimately, it will be up to the user to identify whether or not a particular behavior of Emulab could affect the reliability of her/his results.

The logical topology we have used in all the following experiments consists of a 100Mbps LAN with three user nodes (Fig. 2), one of them connected at 10Mbps. This allows us to experiment with the emulation of both 100Mbps and 10Mbps interfaces. In the case of 10Mbps, hardware limitations in terms of packets per second processing are not very crucial since the injected traffic is much lower.

We have defined three configurations, two of them using emulation strategies and a real one, without emulation, that is used as a reference:

1) Emulation with delay-node-shaping (Fig. 2(a)): in this configuration, which is the default in Emulab, three delay nodes running the Dummynet software are allocated by Emulab in order to model the links which connect the user nodes to the network. Dummynet configures two pipes, inbound and outbound ones, to shape traffic entering and leaving a user node. Inbound and outbound pipes have queues of 5 and 50 slots respectively. Therefore, six experimental nodes are needed to run this configuration.
2) Emulation with end-node-shaping (Fig. 2(b)): this second configuration of Emulab reduces the number of experimental nodes by enabling Dummynet into the user nodes. Therefore, we only need three experimental nodes to run the same experiment, instead of six as in the previous case. The only drawback is that this affects the performance since we are running on the same hardware all the processes, end user and network emulation ones.

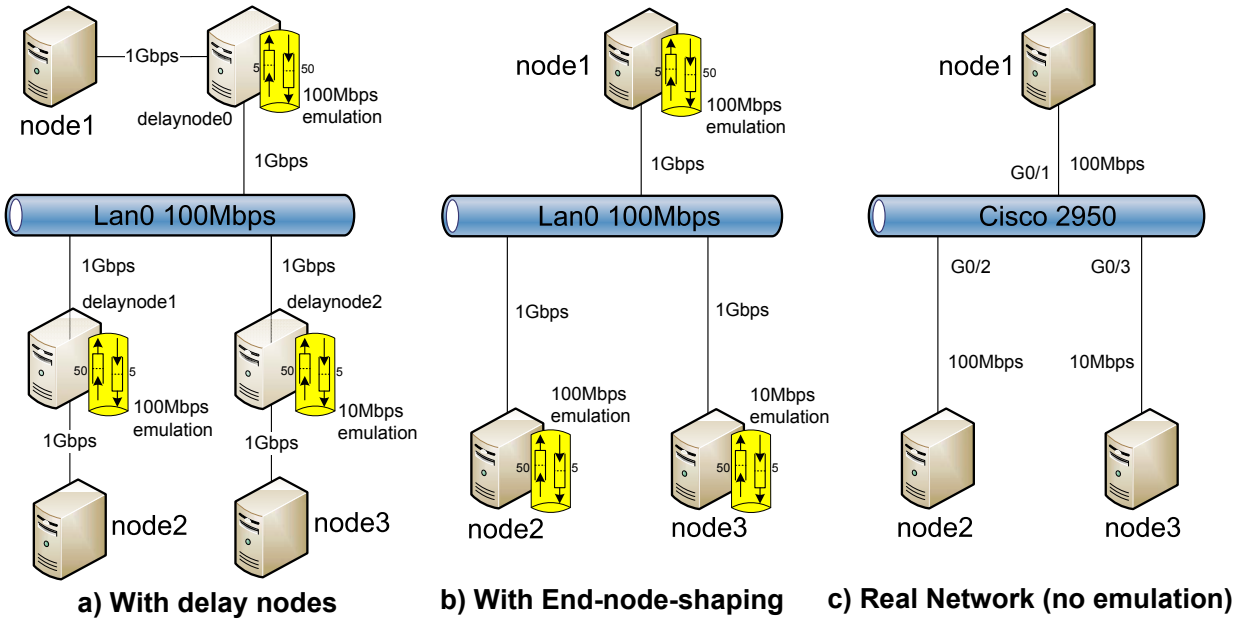


Fig. 2. The three configurations used in our experiments: a) and b) with emulation, c) reference network without emulation.

As for the queuing configuration, it is identical to the first configuration.

3) Reference configuration without emulation (Fig. 2(c)): it consists of a real network with three nodes connected to a Cisco 2950 switch at the same speeds that we are testing in the previous emulated configurations. Neither emulation nor simulation is used in this case.

In most of the experiments all nodes are of the same type in order to avoid that results are altered due to hardware dependencies. They are Dell PCs with AMD 2GHz Athlon processors and 2GB of RAM, running FreeBSD as operating system. The only exception is while testing the repeatability (Section III-C), where we introduce a second type of hardware in order to study the effect of combining different types of hardware. This second one consists of Fujitsu PCs with Intel PIV processors and 1GB of RAM. As for the operating system in the delay-nodes it is always FreeBSD 6.4.

We have used different tools and approaches to collect and to analyze the experimental data, particularly for the measurement accuracy experiments (Section III-D):

- **Iperf** [18] is a tool to generate UDP traffic between a source node and a sink node. But it also includes a built-in measurement functionality that provides statistics such as sustained bandwidth and packet loss, that we use to assess the network performance.
- The **Constant Bit Rate (CBR) traffic generator** supported by Emulab works similar to Iperf. It does not provide statistics, but it is easier to use and schedule in the NS script file as shown in [12].
- **TCPReplay** [19] uses a previously captured traffic file in libpcap format and replays it back into the network, usually to test switches, routers and firewalls. It is a powerful tool that allows to classify traffic as client or server and rewrite Layer 2, 3 and 4 headers.
- **TCPivo** [20] is another, less known, free and open-source tool that supports high-speed packet replay from a trace file.
- **TCPdump** [21] is a well known software that allows to capture and to store packets transmitted and received on a network interface.
- The "**SPAN**" or "**monitor**" feature of Cisco switches [22] together with TCPdump is a less intrusive way to intercept traffic, in the sense that it does not affect the node performance since it is an external process. The switches we use in the experimental plane of the platform allow us to configure two monitor sessions.

In this study we used the aforementioned traffic generators, i.e., Iperf, TCPReplay and TCPivo, in order to produce non-responsive traffic (open-loop), e.g., unidirectional UDP flows. These tools fulfill the needs of our experiments, which are focused on the network emulation capabilities of the Emulab platform, rather than on application protocols. However, researchers might find the need to use other tools, such as Tmix [23] and Swing [24], that can produce responsive traffic (closed-loop).

An important element of scientific analysis is to explore the parameter space with multiple experiments and analyze the collected results with statistical methods. In this sense, we show in Fig. 3 how we can run multiple experiments in an Emulab testbed. The shell script swaps in and out an experiment from the server *ops*, generates traffic with different bandwidth and packet size, and finally stores statistics in a repository for further analysis.

B. Fidelity

We have studied the fidelity of Emulab in terms of network performance and shaping accuracy.

```
#!/bin/csh
# Variables
set MYPROJ=projectX
set MYEXP=experimentX
set LOGFOLDER=/local/logs
set RIPERF=/usr/local/etc/emulab/emulab-iperf
set RSSH="ssh -n -o StrictHostKeyChecking=no"
# Repeat 10 times the experiment
foreach n (1 2 3 4 5 6 7 8 9 10)
  # Swap in experiment EXP from project PROJ
  script_wrapper.py --server=boss swapexp PROJ EXP in
  # And wait until it is active
  script_wrapper.py --server=boss expwait PROJ EXP active
  # Launch server iperf in sink node
  $RSSH nodeX.$MYEXP.$MYPROJ "$RIPERF -s -u
              >>& $LOGFOLDER/nodeX-iperf.log &"
  # Range of bandwidth to generate
  foreach bw (1000000 5000000 10000000 20000000)
    # Range of packet size
    foreach ps (64 128 256 512)
      # Launch client iperf to generate traffic
      $RSSH nodeY.$MYEXP.$MYPROJ
            "$RIPERF -c nodeX -l $ps -t 30 -b $bw -u
            >>& $LOGFOLDER/nodeY-iperf.log"
    end
  end
  # Syncing & Archiving
  loghole -e PROJ/EXP sync
  loghole -e PROJ/EXP archive
  loghole -e PROJ/EXP clean -n -f
  # Swap out the experiment
  script_wrapper.py --server=boss swapexp PROJ EXP out
  # And wait until the process has ended
  script_wrapper.py --server=boss expwait PROJ EXP swapped
end
```

Fig. 3. Method of conducting multiple experiments in order to explore the parameter space (several combinations of bandwidth and packet sizes).

1) Network performance: In order to study the network performance, i.e., received traffic by the sink node versus sent traffic from the source node in the cases of 100Mbps and 10Mbps LAN emulation, we generate UDP traffic with Iperf from node1, first to node2 and then to node3. We run multiple experiments through a script as shown in Fig. 3, varying the packet size from 64 bytes up to 1408 bytes, and the generated traffic from 0Mbps up to either 100Mbps or 10Mbps depending on whether the sink node is connected at 100Mbps or 10Mbps. Fig. 4, 5 and 6 depict the experienced network performance for the reference scenario and the two emulation strategies. The figures (a) and (b) correspond to the two different network speeds.

The ideal performance would be a diagonal straight line (y=x) reaching up to the nominal interface speed and followed by a horizontal line. In reality, the behavior of the configuration without emulation (Fig. 4) is close to the ideal one, at least for packets larger than 256 bytes. Even in the worst case, of 64 byte packets, the performance reaches 80% of the interface speed. Above this point, the switch is not able to handle all the packets and there are drops before reaching the maximum speed.

Conversely, if we look at the emulated configurations, with a delay-node-shaping strategy (Fig. 5) we can see that the performance depends significantly on the packet size, specially in the 100Mbps case. However, we still see the same behavior (curve shape). For each packet size the performance follows the y=x line up to a saturation point and then it becomes horizontal. In the 10Mbps case the situation is better and the performance reaches the interface speed for packets larger than 128 bytes. In any case, the figures confirm the expected result because the smaller the packet is, the higher the number of generated packets is, given a fixed bandwidth. This implies that the hardware and software components of the topologies need to handle more packets and they reach the processing limit earlier. This limit is due to inherent hardware and software limitations in terms of packets per second and interrupts per second processing as explained in [25].

On the other hand, in the end-node-shaping configuration (Fig. 6) the behavior differs from the expected one. As the traffic grows, the performance seems to follow the behavior mentioned earlier, but at a certain point, the curves divert and tend again to the y=x diagonal line. This means that Dummynet fails to model the network and exhibits an unstable behavior, which is attributed to the double role of the user nodes, generating and processing traffic (Iperf) as well as network emulation with Dummynet.

Based on these results, we are able to confirm that experimentation with Emulab provides realistic results in terms of trend and behavior when using delay-node-shaping configuration. Obviously, we cannot compare the results in absolute terms because the values of our metric are very hardware-dependent. In fact, we can expect better results in terms of performance by using more powerful nodes in the emulation platform, as well as worst results by using a less powerful switch in the real scenario, but in all cases, with analogous behavior.

End-node-shaping strategy might still be useful in the case of experiments with low bandwidth utilization (0-30%) and limited number of experimental resources.

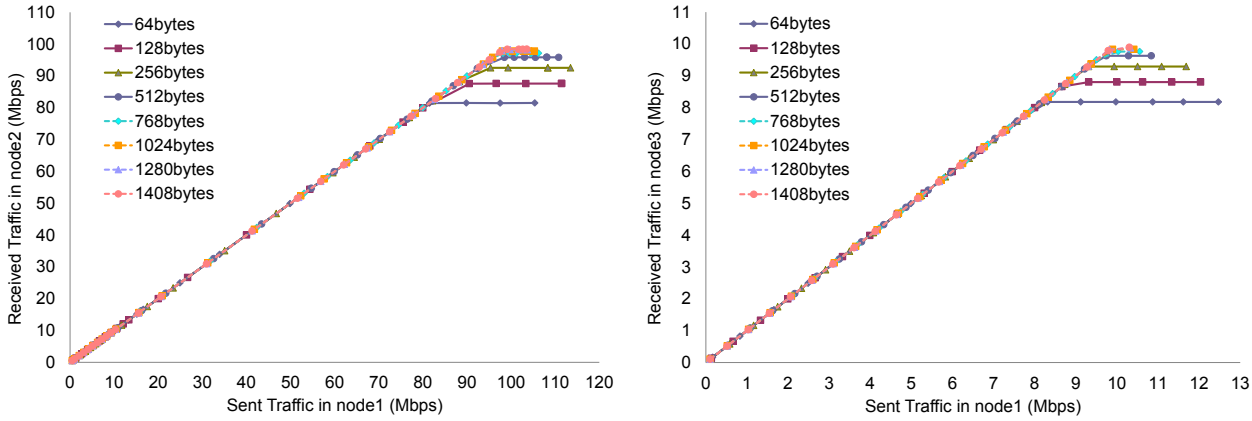
> ***Guideline 1: Emulab with delay-node-shaping strategy can realistically recreate with acceptable quality the performance of real networks, while the end-node-shaping strategy is reliable only in the case of low bandwidth utilization (0-30%).***

2) Shaping accuracy: There are two main configuration parameters in Dummynet that affect its task as traffic shaper, namely queue size and delay. In this section we discuss the bandwidth shaping accuracy of Dummynet with respect to these two parameters in the delay-node-shaping configuration (Fig. 2(a)). Since the inbound queue in Dummynet is significantly smaller than the outbound (5 versus 50), the inbound queue is the most restrictive and the place where most of the drops occur. We should underline that all interfaces in our experiments are physically configured at 1Gbps and it is up to Dummynet to shape the traffic at the desired speed. Thus, it is possible to measure more than 100Mbps in some interfaces.

In order to configure a 100Mbps Dummynet pipe with a queue size of 10 slots, we can perform the following remote action in the delay nodes from the script given in Fig. 3:

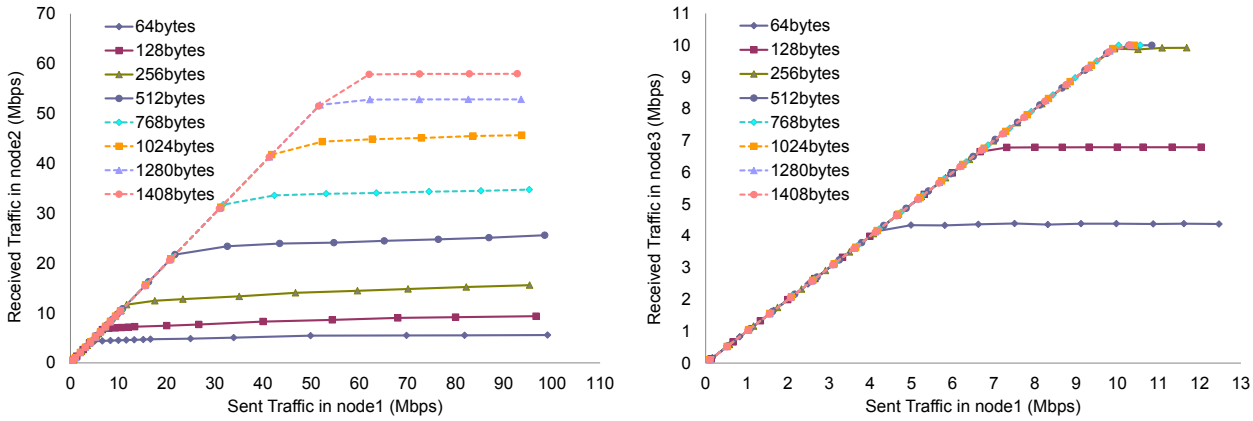
```
"sudo ipfw pipe pipe_number config bw 100Mbit/s queue 10"
```

We launched a set of experiments where we generated more than 140Mbps of CBR traffic with large packets from node1, first to node2 and then to node3 with different inbound queue sizes and delay-node-shaping configuration. Fig. 7 shows the



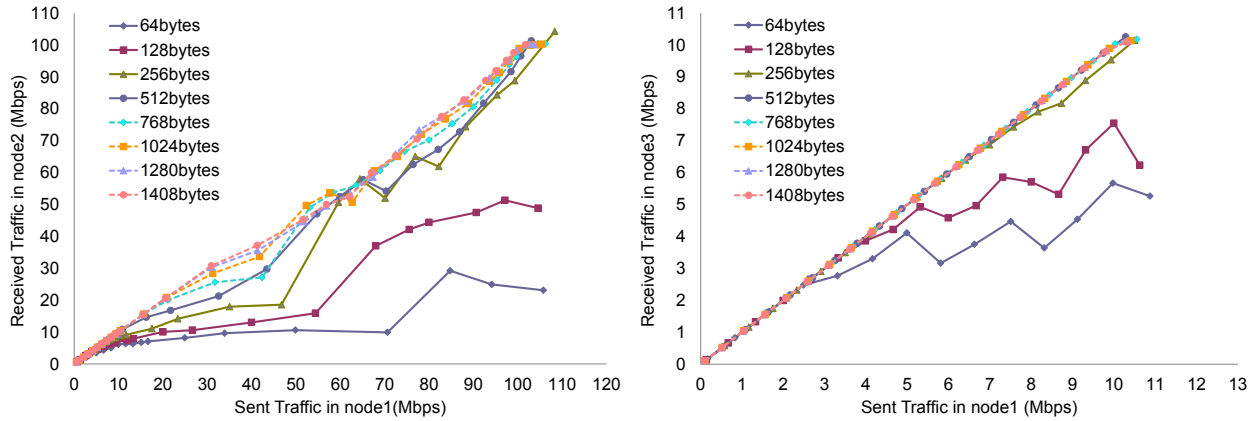
(a) traffic toward 100Mbps node (b) traffic toward 10Mbps node

Fig. 4. Network performance in the reference configuration without emulation.



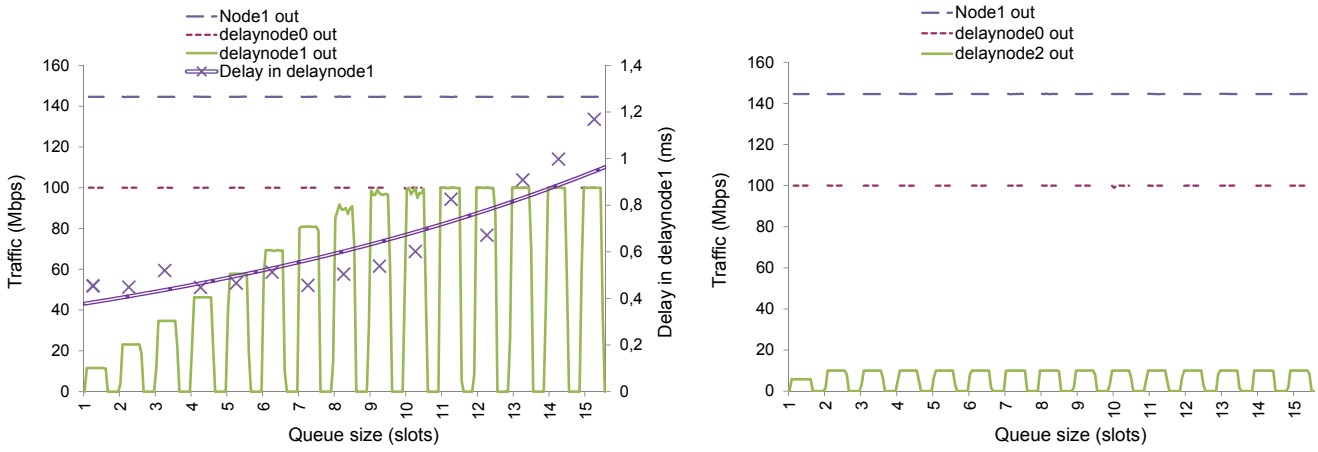
(a) traffic toward 100Mbps node (b) traffic toward 10Mbps node

Fig. 5. Network performance in the emulation configuration using delay-node-shaping.



(a) traffic toward 100Mbps node (b) traffic toward 10Mbps node

Fig. 6. Network performance in the emulation configuration using end-node-shaping.



(a) traffic toward 100Mbps node

(b) traffic toward 10Mbps node

Fig. 7. Influence of Dummunet queue size in shaping performance.

results for 100Mbps and 10Mbps shaping, where each pulse corresponds to a different inbound queue size, from 1 to 15 slots. In this figure three curves are visible: a) the dashed curve depicts the traffic leaving node1, more than 140Mbps; b) the dotted curve corresponds to traffic leaving delaynode0, where the outbound speed of 100Mbps for the node1 is modeled; and c) the solid curve shows traffic leaving delaynode1 or delaynode2, depending on the case, where the inbound speed is modeled for the sink nodes.

In the reference configuration there are no delay nodes and the user nodes have their interfaces connected at the respective speed of 10Mbps and 100Mbps. In this case we only see 100Mbps of traffic leaving node1 and entering node2, and 10Mbps entering node3.

The results of the emulated configuration show that there is a clear impact of the queue size in the shaping accuracy for the 100Mbps case, where the queue size has to be 11 slots or larger to reach the desired speed. On the contrary, for the 10Mbps case, this speed is already reached with 2 slots in the queue size.

Although this is a useful observation, the user should be careful before configuring a larger queue size since it might affect the network delay, which depends on the following variables:

- Delay-node processing time is the most influencing variable and depends on Dummynet's queue size and the hardware processing power.
- Switch processing time is insignificant compared to the delay-node processing time.
- Transmission time is insignificant compared to the delay-node processing time, considering that all the interfaces are configured at 1Gbps.
- Propagation delay is minimal in a LAN configuration.

In order to measure the impact of the first variable in the network delay, we captured packets entering and leaving delaynode1 in the previous experiments. In Fig. 7(a), the double curve is an exponential trend line of the average delay in delaynode1 for the different queue sizes. It clearly shows that the delay increases as the queue size does, although it is only after 8 slots when it does it significantly. In the case of 11 slots, where a good shaping accuracy is reached, the delay increases 0.37ms compared to the case of 2 slots. This delay could be multiplied in larger topologies if the packets traverse many delaynodes with large queue sizes.

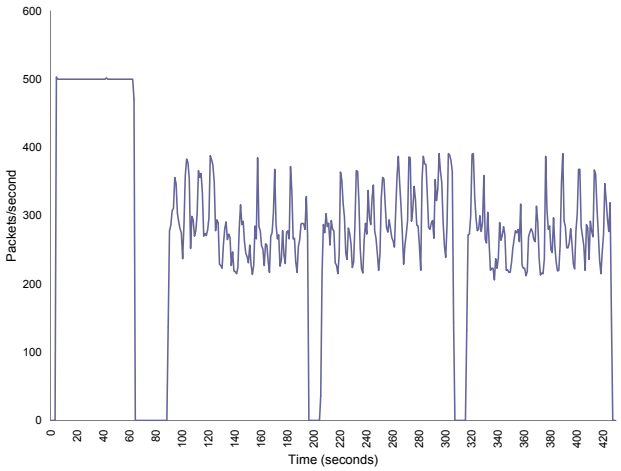
Another important observation is that configuring a different delay in Dummynet, instead of 0ms, does not change the results in terms of shaping accuracy.

> ***Guideline 2: Accurate shaping is hard to achieve under high load conditions without introducing network delay ($<$ 0.5ms) due to processing. Depending on the goal of the experiment and the size of the network, the inbound queue size of Dummynet should be changed from its default value.***

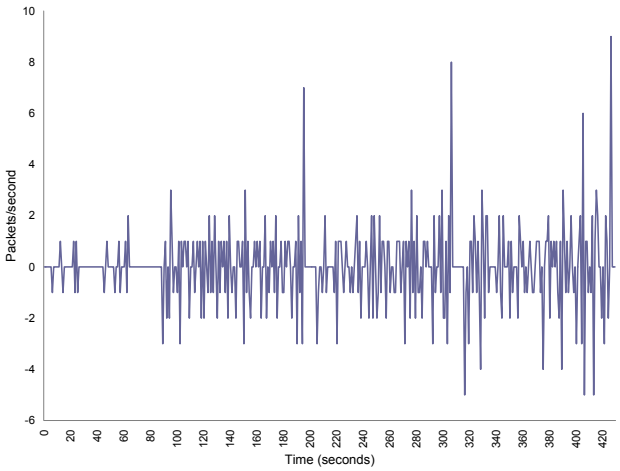
In addition, we tested whether there was unnecessary shaping of traffic signals with bandwidth lower than the nominal bandwidth of the emulated links. In order to do that, in the delay-node-shaping configuration (Fig. 2(a)), we generated traffic, $gen(t)$, from node1 towards node2, and specifically a 5 Mbps CBR pulse followed by a sample of real traffic (taken from the DATCAT repository [26]) injected with TCPReplay. The traffic passes through delaynode0 and delaynode1 before reaching the sink node2. Fig. 8 shows the measured packet/second in the source node in a period of around 3 minutes as well as the difference between the generated and the received traffic signal $rec(t)$. The results show that the generated signal in node1 travels through the network keeping its bursty nature and integrity; the small differences (less than 3%) are attributed to the artifacts of time correlation and statistics gathering processes. In fact, if we calculate the sum of all the values in this figure, we get 0, so all the packets are traversing the network from node1 to node2.

C. Repeatability

Scientifically rigorous experimentation does not only mean to get realistic performance, but being able to reproduce results at any time, even by a different researcher with the same tools and configuration. Therefore, we have carried out

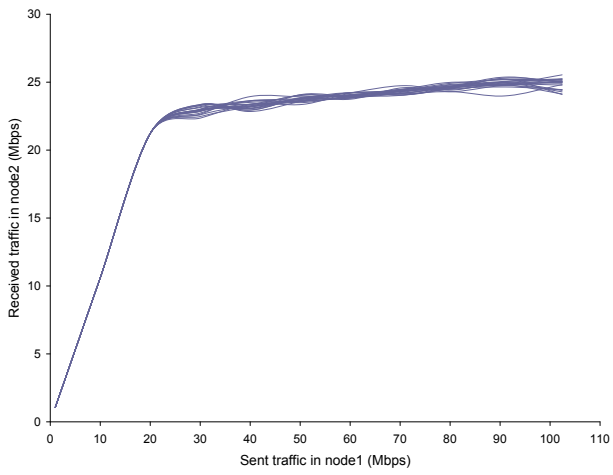


(a) traffic signal $gen(t)$ as generated in node1

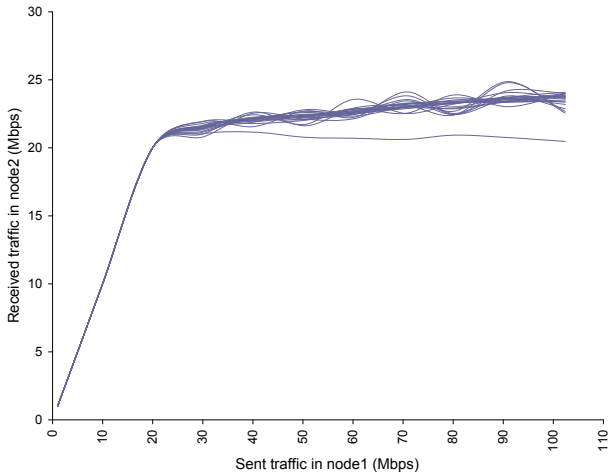


(b) difference between generated and received signal in node2 $gen(t) - rec(t)$

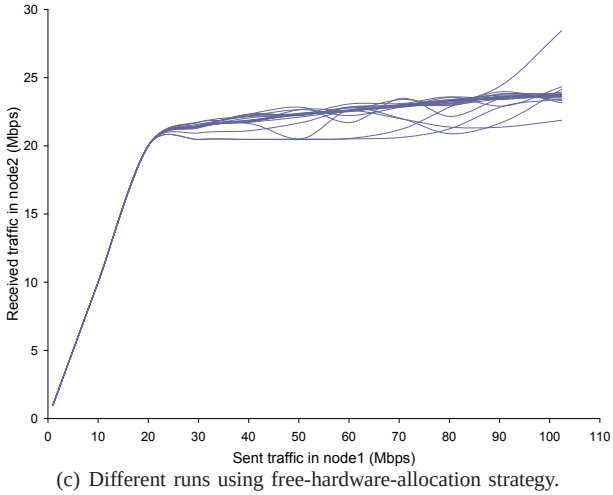
Fig. 8. Testing traffic signal propagation fidelity.



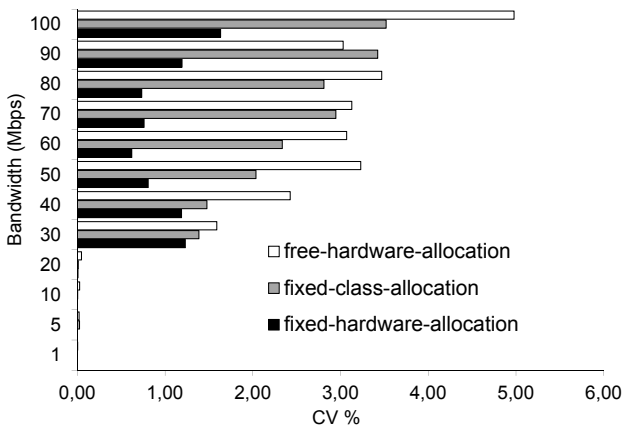
(a) Different runs using fixed-hardware-allocation strategy.



(b) Different runs using fixed-class-allocation strategy.



(c) Different runs using free-hardware-allocation strategy.



(d) Coefficient of variation for the tree scenarios of hardware allocation.

Fig. 9. Repeatability of network performance in three different strategies of hardware allocation.

appropriate tests to study the impact of hardware allocation strategies, events generation system and traffic generator tools to repeatability.

1) Hardware allocation strategies: As we have seen earlier in Section III-B the quantitative results of an Emulab-based experiment are hardware-dependent. On the other hand, the hardware allocation might change from one experiment to another due to (un)availability of resources, randomness in Emulab's swap-in algorithm [27] or other testbed policies. For this reason, we have performed a set of experiments in order to study the influence of hardware allocation on the repeatability of an experiment in the delay-node-shaping configuration (Fig. 2(a)). We measured the network performance, i.e., the traffic received by the sink node versus the traffic sent by the source node, along repetitive experiments for three hardware-allocation strategies:

1) the *fixed-hardware-allocation strategy*, where each experimental node is fixed to a specific PC;
2) the *fixed-class-allocation strategy*, where experimental nodes are chosen from the class of Dell PCs;
3) the *free-hardware-allocation strategy*, where experimental nodes are freely chosen from the two hardware classes, i.e., Dell and Fujitsu PCs. This is the default behavior in Emulab.

The following NS code creates three nodes: the first one is allocated in a fixed PC, the second one in a PC belonging to a specific class (dell), and the last one is freely allocated (default configuration).

```
set node1 [$ns node]
tb-fix-node $node1 pc20

set node2 [$ns node]
tb-set-hardware $node2 dell

set node3 [$ns node]
```

In all experiments, we measured the sustained network performance using Iperf's built-in measurement functionality. We generated UDP traffic from node1 to node2 with 512 bytes of payload and bandwidth ranging from 0Mbps up to 100Mbps. In the first experiment set we did not swap in and out in order to preserve the exact hardware allocation (not even a change in delay nodes[1]), while in the other two experiment sets we swapped in and out for each experiment, leaving Emulab to freely choose the hardware allocation according to the predefined strategy.

For each experiment set, we run the same experiment 20 times and the results are depicted in Fig. 9(a), 9(b) and 9(c), while the statistics, average μ, standard deviation σ and coefficient of variation ($CV = \frac{\sigma}{\mu}$), are shown in Fig. 9(d). From the figures we can see that the 20 experiments in each case provide the same performance when traffic is under 20Mbps, i.e., when there are no packet losses, the experiments are accurately repeatable. However, when Dummynet is not able to process all the packets and there are drops in the queues, we see that the hardware allocation introduces a higher variability in the results as we go from a fixed to a free allocation strategy.

[1]The Emulab software does not allow the user to fix the hardware allocation of delay nodes.

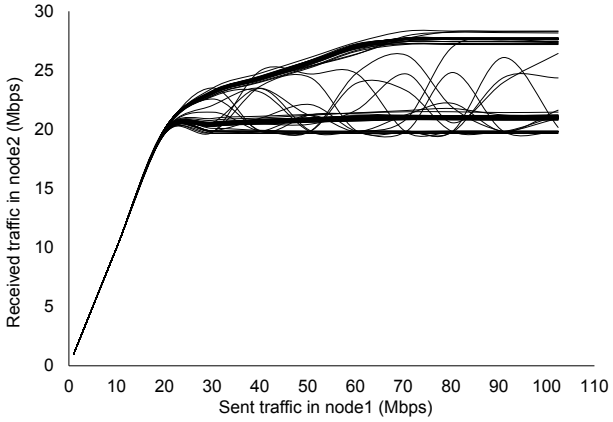


Fig. 10. Different runs using free-hardware-allocation strategy in the DETER.

In fact, if we look at the CV, which in general gets worse as the bandwidth grows, the best results in terms of repeatability are with the fixed-hardware-allocation strategy and the worst results with the free-hardware-allocation strategy. Another important observation is that even in the worst case the CV is under 5%, i.e., the maximum CV for the three allocation strategies is 1.63%, 3.52% an 4.98% respectively.

2) Repeatability across testbeds: Finally, we have run the same sets of experiments in DETER [7], which is an Emulab based public facility, in order to test whether we can get similar results using an equivalent testbed. For both fixed-hardware and fixed-class allocation strategies, the results are of the same order while in the third strategy, free-hardware-allocation, the CV goes up to 14% with high network load (Fig. 10). The reason behind this is that DETER is a larger testbed with more than ten PC classes while our facility only consist of two PC classes. Therefore, it is important to keep in mind that there is a hardware dependency in the results, specially when working in heterogeneous environments, that might lead to non repeatable results.

Guideline 3: Emulab provides accurate repeatable results in experiments with moderate network load regardless of the hardware allocation strategy and the platform. However, as the network load grows and provokes packet drops, the fixed-hardware-allocation strategy is the only way to get an acceptable level of repeatability.

3) The events generation system: In this section we study the accuracy of the events generation system in Emulab. In order to do that, we have run the same experiment 20 times (swapping in and out) using the delay-node-shaping configuration (Fig. 2(a)). In the NS script of the experiment we scheduled four events to generate 2 pulses of CBR traffic from node1 to node2 (see NS code below). Each pulse (events A and C) had a duration of 5 seconds, and the time between them (event B) was of 20 seconds (Fig. 11). The information registered by Emulab about events generation in each experiment is precise and consistent with the configuration.

```
set cbr12 [new Application/Traffic/CBR]
$cbr12 set interval_ 0.003
$cbr12 set packetSize_ 512

set udp12 [new Agent/UDP]
set udpsink12 [new Agent/Null]

$cbr12 attach-agent $udp12

$ns attach-agent $node1 $udp12
$ns attach-agent $node2 $udpsink12

$ns connect $udp12 $udpsink12

$ns at 10.0 "$link0 trace start"
$ns at 20.0 "$cbr34 start"
$ns at 25.0 "$cbr34 stop"
$ns at 45.0 "$cbr34 start"
$ns at 50.0 "$cbr34 stop"
$ns at 60.0 "$link0 trace stop"
```

We captured the traffic with TCPdump in delaynode0 and we measured the time between the first and last packet of each pulse for the 20 experiments. Table I shows the statistics of the duration of events A, B and C along the 20 experiment runs. The standard deviation is always below 63 milliseconds, which should be precise enough for most experiments that schedule events in seconds or tenths of seconds but not necessarily for all experiments, e.g., in Industrial Networks the control systems using SCADA protocols might be sensitive to delays in the order of 10-100 milliseconds. In terms of CV, we see that the accuracy is better with longer periods.

This variation can be explained by looking at what happens after the events are scheduled by the system and before the traffic is captured. The events imply starting or stopping an application (CBR traffic generator) in a remote node, so there is a communication between the Emulab system and node1. Then, the traffic arriving to the delaynode0 has to pass through network cards, switch and cables before it is captured. All these processes, along with CPU scheduling inaccuracy, lead to time shifting. The conclusion is though that Emulab's events generation system is accurate and consistent.

On the other hand, we ran the same experiment with different load conditions in *boss*, the Emulab server in charge of managing the experiments and the events system. We used a script to stress boss, as if many users were responsible for it. Fig. 12 shows the CPU load in *boss* during a set of 9 experiments that test the precision of events generation, going from 0% up to 99%. The duration of events A, B and C are always within twice the standard deviation shown in Table I and without a particular trend as the CPU load increases.

Guideline 4: Emulab's events generation system can be safely used with a precision of tenths of seconds.

TABLE I
THE AVERAGE, STANDARD DEVIATION AND COEFFICIENT OF VARIATION OF THE DURATION OF EVENTS A, B, C.

Event	Avg (sec.)	Stdev (sec.)	CV (%)
A	4.94	0.03	0.55%
B	20.08	0.05	0.25%
C	4.86	0.06	1.30%

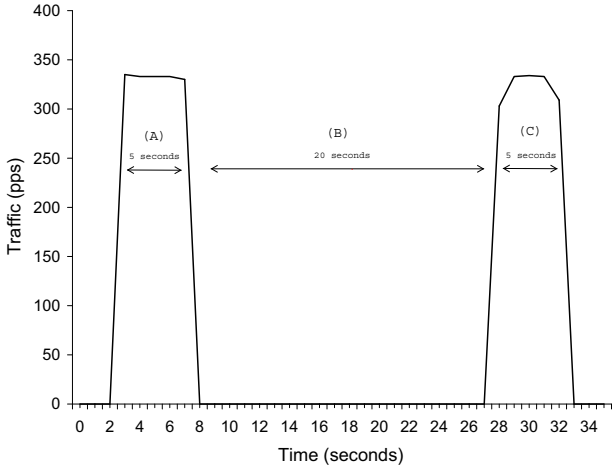


Fig. 11. CBR traffic scheduled in the NS script to be sent in node1.

4) Traffic generators: In this part of the study, we have analyzed the repeatability of different traffic generators, namely Iperf, CBR, TCPReplay and TCPivo, by running each of them 10 times using the delay-node-shaping configuration (Fig. 2(a)). For Iperf and CBR we generated pulses of 30 seconds with UDP packets of 512bytes of payload from node1 to node2 (synthetic traffic), while for TCPReplay and TCPivo, we reproduced a real trace of 30 seconds with TCP packets of random length (taken from the DATCAT repository [26]).

TABLE II
REPEATABILITY OF TRAFFIC GENERATORS (TRAFFIC DURATION).

Tool	Avg (sec.)	Stdev (sec.)	CV (%)
TCPReplay	48.45	3.27	6.74%
TCPivo	30.00	0.00	0.00%
CBR	30.07	0.02	0.05%
Iperf	30.00	0.00	0.00%

Fig. 13 shows the traffic measured with TCPdump in node2 for each of the traffic generators. At first sight, we see that TCPReplay is not able to reproduce a single trace with the same characteristics, each reproduction is different from the other. On the other hand, the rest of the tools seem to generate traffic in a repeatable way. In fact, if we look at Table II, the duration of traffic is practically the same for all the tools but TCPReplay, where the CV is higher than 6% and the standard deviation is 3.27 seconds.

Furthermore, we subtracted the generated traffic signal that was produced by TCPReplay and TCPivo as measured in node2, i.e., $gen(t)$, from the original reference signal of the trace file that we used as input for both tools, i.e., $ref(t)$. We depict the difference $ref(t) - gen(t)$ in Fig. 14. We see that the differences are in the order of a few Kbps in the case of TCPivo, but they increase up to 2Mbps with TCPReplay. A small difference was expected due to the buffering mechanism in the network, but TCPReplay was not able to provide repeatable results.

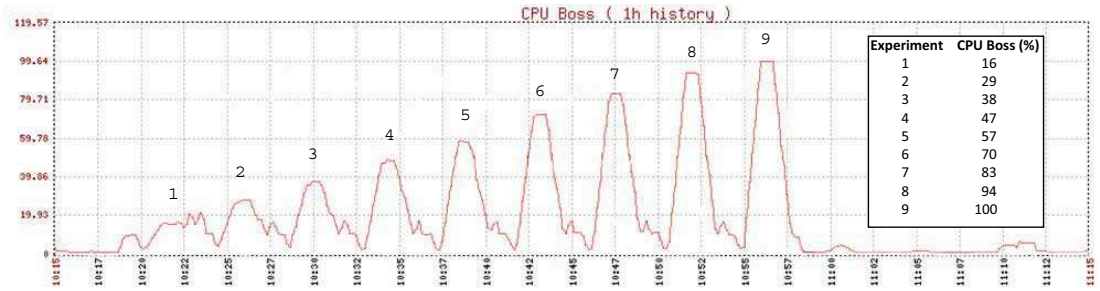
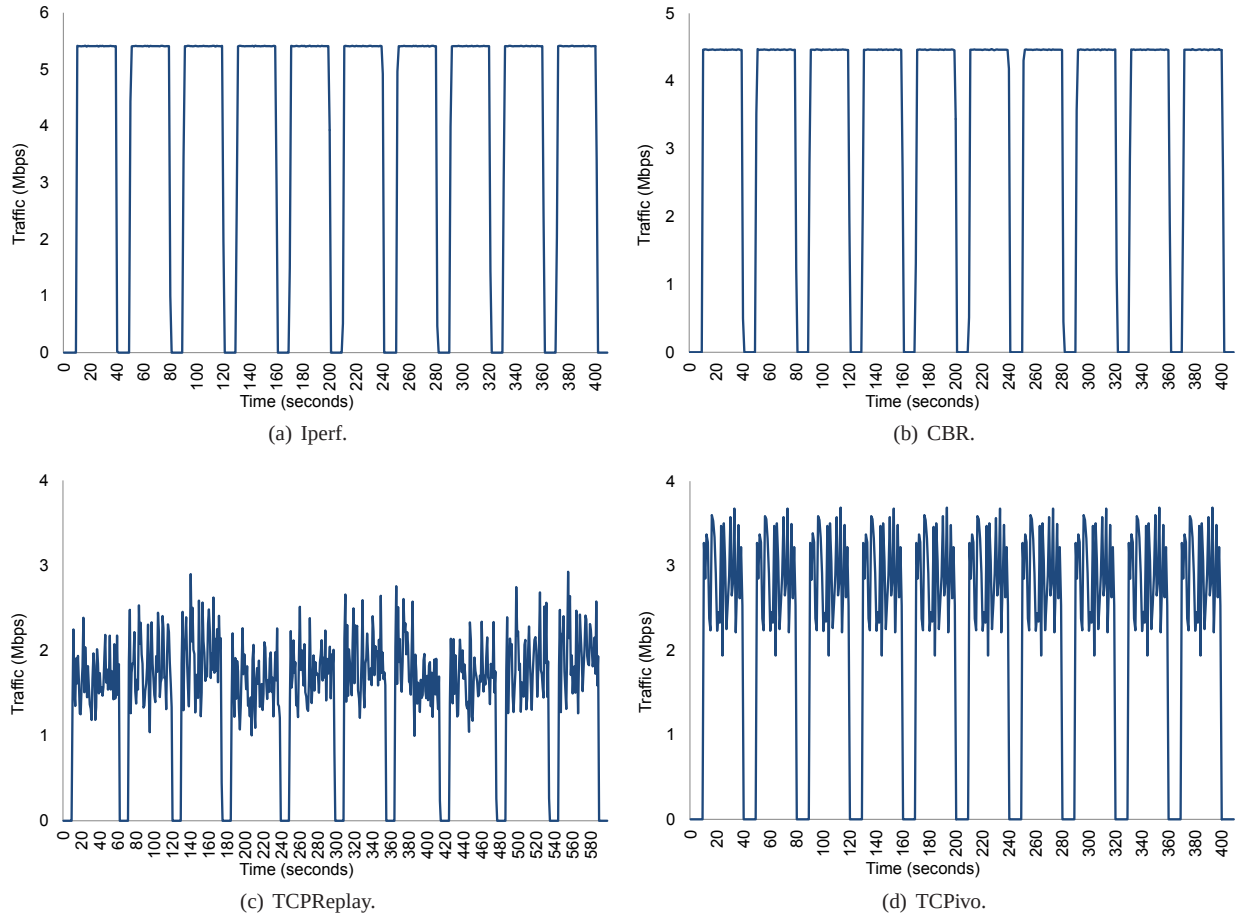


Fig. 12. CPU load in *boss* during the event generation system evaluation.



(a) Iperf.

(b) CBR.

(c) TCPReplay.

(d) TCPivo.

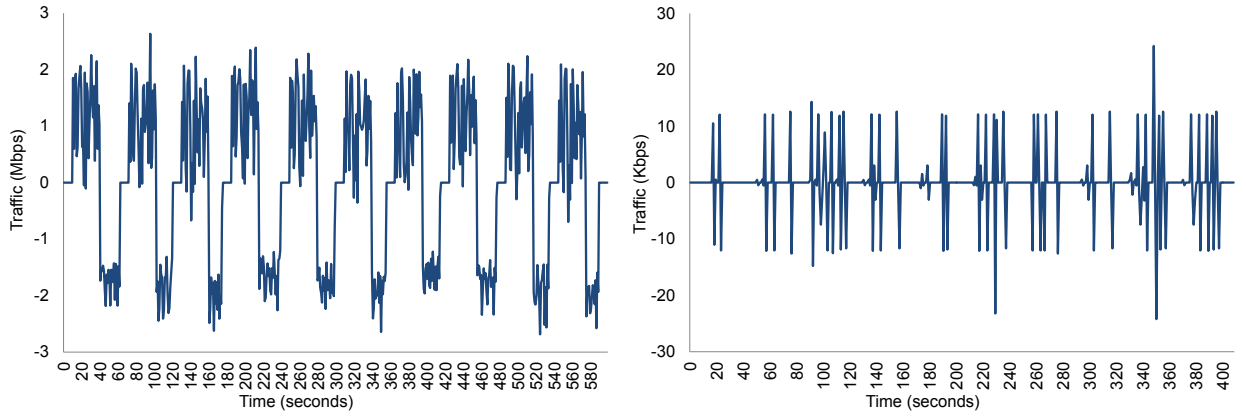
Fig. 13. Traffic generated by different tools and measured in the sink node.

Guideline 5: In order to quantitatively reproduce the same experimental results the researcher is encouraged to use reliable traffic generators, e.g., TCPivo.

D. Measurement accuracy and interference

In this part of the study we aimed to validate the different measurement tools we used in our experiments. Therefore we ran a single experiment where we generated traffic with Iperf from the source node1 to both sink nodes, node2 and node3, using the delay-node-shaping configuration (Fig. 2(a)). We gathered the traffic statistics provided by different tools (Dummynet, Iperf, TCPdump) and compared them to the measurements that were yielded by an external measurement process, i.e., by spanning network traffic to a dedicated measurement node, that acted as reference.

All tools, as well as the reference, provided exactly the same information. This implies that they are valid measurement tools and provide accurate information. However, as shown in Table III, each of the tested tools has its pros and cons in terms of provided information, scalability and processor consuming. For example Spanning traffic provides precise and complete information about every single packet in the network without affecting at all an experiment, but its use is limited to the switch limitations.



(a) $ref(t) - gen(t)$ for TCPReplay (b) $ref(t) - gen(t)$ for TCPivo

Fig. 14. Subtraction between traffic signals of the reference trace file $ref(t)$ and the generated traffic $gen(t)$ by TCPReplay and TCPivo.

TABLE III
ADVANTAGES AND DISADVANTAGES OF DIFFERENT MEASUREMENT APPROACHES

Measurement tool	Advantage	Disadvantage
Dummynet	Lightweight	Only aggregate statistics
Iperf	Lightweight	Application specific, aggregate statistics
TCPdump in delay-node	Very detailed	Heavy, might cause interference
Spanning traffic	Very detailed, no interference	Not scalable (switch limitations)

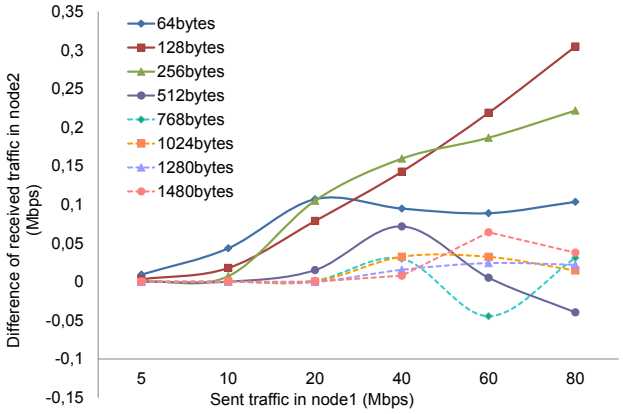


Fig. 15. Difference in the performance between experiments with & without TCPdump.

Guideline 6: A researcher should be aware of the different measurement approaches, that although accurate, have significant advantages and disadvantages (Table III).

On the other hand, it is well known that measuring any parameter in a system may interfere and change its behavior [28]. We have already seen that some results are hardware dependent, so adding monitoring processes to the nodes, such as TCPdump, can interfere with the system behavior.

In order to study whether this interference is significant or not, we analyzed whether the network performance is affected when TCPdump is running in the delay-nodes. We ran multiple experiments, varying the packet size from 64 bytes up to 1408 bytes, and the generated bandwidth from 5Mbps up to 80Mbps of payload. For each combination and for the two scenarios (with and without TCPdump running in the delay-nodes) we ran 100 experiments. Fig. 15 depicts the subtraction of the average network performances between the two scenarios when traffic is sent toward node2 (100Mbps) and for different packet sizes. Positive values mean better performance when TCPdump is not running.

Apparently, the performance is better with smaller packet sizes when TCPdump is not running, whilst for packet sizes above 512bytes the difference fluctuates around 0. As we could see in the repeatability experiments, part of the variability is attributed to statistical variation. Therefore, a more complete statistical analysis is needed to assess the measurement interference and to make clear whether there is an interference or it is just a statistical variation. In order to do that, we use the Kolmogorov–Smirnov test (KS test) [29].

The KS test allows to compare two samples and to determine if they differ significantly. It uses the maximum vertical deviation between the cumulative distribution functions (CDF) of both samples:

$$max(|F1(x) - F2(x)|),$$

where F1(x) and F2(x) are the CDF of samples 1, and 2 respectively. The null hypothesis is that the samples are from the same continuous distribution. The result of the test, *h*, is 1 if the test rejects the null hypothesis at the 5% significance level, and is 0 otherwise.

In our case, we compared samples of network performance with and without TCPdump running in the delay-nodes. Table IV shows the result for the cases under study. The null hypothesis is rejected with packets of 256 bytes and smaller at any speed, and with larger packets (1280 and 1408 bytes) at high speed (above 60Mbps of payload). On the other hand, we can draw a diagonal, indicated with bold text in the table, which corresponds to cases where there are few dropped packets (1-10 drops/second). Here, the statistic analysis might fail because the measured standard deviation of the samples is very high. We can identify an area with large packets and

TABLE IV
RESULT H OF THE K-S TEST WHILE COMPARING THE DROPPED PACKETS IN THE SCENARIOS OF RUNNING AND NOT RUNNING TCPDUMP IN DELAY NODES

Packet size (bytes)	64	128	256	512	768	1024	1280	1408
5 Mbps	1	1	**1**	0	0	0	0	0
10 Mbps	1	1	**1**	0	0	0	0	0
20 Mbps	1	1	1	**1**	**1**	0	0	0
40 Mbps	1	1	1	0	0	**1**	**1**	**1**
60 Mbps	1	1	1	0	0	0	1	1
80 Mbps	1	1	1	0	0	0	1	1

low bandwidth where the hypothesis is not rejected. This means that both samples come from the same distribution and therefore differences in the network performance are minimal and due to statistical reasons, not due to the interference of TCPdump.

Finally, figure 16 shows a visual representation of the K-S test by depicting the CDF of measured dropped packets (x) when TCPdump is running (F2) and is not running (F1) in the delay nodes for the particular case of 60Mbps and different packet sizes. Once again, we see how the curves get distant with the smaller packet sizes.

Guideline 7: In an experiment where even a small percentage of packet loss increase is significant, traffic monitoring in delay-nodes should be avoided with high network load (20Mbps or higher) or small packets (256 bytes or smaller).

IV. CONCLUSION

Testing and evaluating the resilience of complex systems and networks is a hard process. The limitations of software simulators and the required cost and effort to setup and maintain ad-hoc testbeds of real systems make the use of emulation testbeds a promising approach. Emulation testbeds like Emulab have been recently used for conducting security and resilience research on IP networks. In this paper we investigate several characteristics of an Emulab testbed that are needed for conducting scientifically rigorous experiments, and specifically experimental fidelity and repeatability as well as measurement accuracy and interference.

Our contribution can be summarized in the following points. We confirm that the current trend of using emulation testbeds is justified as both realistic and efficient. Nevertheless we highlight the fact that the interpretation of experimental results should not be based on absolute numbers (which are highly hardware dependent) but rather on system behavior and trends, except when using the same or very similar hardware. This means that Emulab-based configurations are representative of real systems in terms of emerging behavior (qualitative) rather than absolute performance (quantitative). Repeatability can be achieved and the platform allows the use of several accurate and complementary measurement approaches. As a concluding remark, we can state that Emulab-based testbeds, such as EPIC and DETER, are promising scientific tools for the study of security and resilience of ICT systems mainly because of their system representativeness (fidelity) under extreme conditions and failures but in addition due to their automation and their ability to consistently reproduce experimental results. In this context we formulate a set of guidelines that future Emulab users should consider while designing their experiments or analyzing their results.

Further work in the direction of evaluating Emulab as a tool for emulating complex systems is needed, for example a study of the artifacts introduced by the use of virtualization, as well as towards the improvement of the link emulation mechanisms, e.g., by replacing Dummynet with a better alternative [30].

REFERENCES

[1] G. Huston, M. Rossi, and G. Armitage, "Securing BGP – a literature survey," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 2, pp. 199–222, quarter 2011.

[2] B. Genge and C. Siaterlis, "Developing cyber-physical experimental capabilities for the security analysis of the future Smart Grid," in *Proceedings of the 2011 IEEE Innovative Smart Grid Technologies*, 2011, pp. 1–7.

[3] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid – the new and improved power grid: A survey," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–37, 2011.

[4] Emulab, "Emulab bibliography," http://www.emulab.net/expubs.php/, 2012, [Online; accessed March 2012].

[5] C. Siaterlis and M. Masera, "A survey of software tools for the creation of networked testbeds," *International Journal On Advances in Security*, vol. 4, no. 1-2, pp. 1–12, 2010, ISSN. 1942-2636.

[6] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255–270, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844152

[7] DETER, "DETER. cyber-DEfense Technology Experimental Research laboratory Testbed," http://www.isi.edu/deter/, 2012, [Online; accessed March 2012].

[8] J. Mirkovic, A. Hussain, S. Fahmy, P. Reiher, and R. Thomas, "Accurately measuring denial of service in simulation and testbed experiments," *Dependable and Secure Computing, IEEE Transactions on*, vol. 6, no. 2, pp. 81–95, april-june 2009.

[9] D. S. Anderson, M. Hibler, L. Stoller, T. Stack, and J. Lepreau, "Automatic online validation of network configuration in the emulab network testbed," in *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, ser. ICAC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 134–142. [Online]. Available: http://dx.doi.org/10.1109/ICAC.2006.1662391

[10] R. Chertov, S. Fahmy, and N. B. Shroff, "Fidelity of network simulation and emulation: A case study of tcp-targeted denial of service attacks," *ACM Trans. Model. Comput. Simul.*, vol. 19, no. 1, pp. 4:1–4:29, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1456645.1456649

[11] Emulab, "Emulab tutorial," https://users.emulab.net/trac/emulab/wiki/Tutorial, 2008, [Online; accessed March 2012].

[12] ——, "Emulab tutorial - a more advanced example," https://users.emulab.net/trac/emulab/wiki/AdvancedExample, 2010, [Online; accessed March 2012].

[13] B. Genge, I. N. Fovino, C. Siaterlis, and M. Masera, "Analyzing cyber-physical attacks on networked industrial control systems," in *Critical Infrastructure Protection*, 2011, pp. 167–183.

[14] ISI, "The network simulator - ns-2," http://www.isi.edu/nsnam/ns/, 2012, [Online; accessed March 2012].

[15] Emulab, "Emulab - testbed ns command extensions," http://users.emulab.net/trac/emulab/wiki/nscommands, 2012, [Online; accessed March 2012].

[16] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 1, pp. 31–41, Jan. 1997. [Online]. Available: http://doi.acm.org/10.1145/251007.251012

[17] A. Garcia, C. Siaterlis, and M. Masera, "Testing the fidelity of an emulab testbed," in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, june 2010, pp. 307–312.

[18] NLANR/DAST, "Iperf: The TCP/UDP bandwidth measurement tool," http://sourceforge.net/projects/iperf/, 2012, [Online; accessed March 2012].

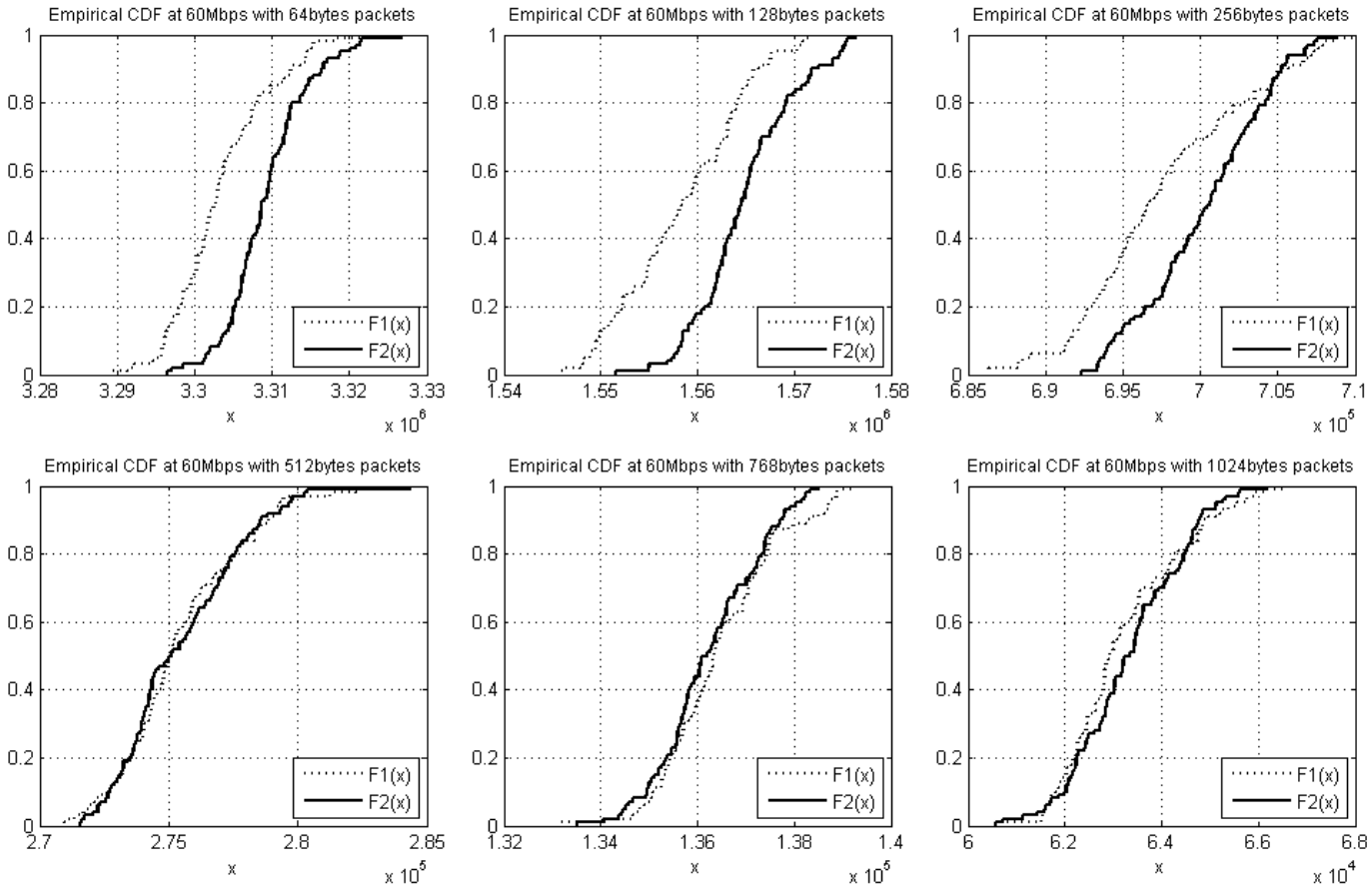


Fig. 16. Cumulative distribution function of measured dropped packets (x) when TCPdump is running (F2) and is not running (F1) in the delay nodes.

[19] A. Turner, "TCPReplay tool," http://tcpreplay.synfin.net/trac/, 2012, [Online; accessed March 2012].

[20] W. chang Feng, A. Goel, A. Bezzaz, W. chi Feng, and J. Walpole, "Tcpivo: A high-performance packet replay engine," in *In Proceedings of the ACM SIGCOMM workshop on Models, methods*, 2003, pp. 57–64.

[21] "Tcpdump: Traffic analyzer," http://tcpreplay.synfin.net/trac/, 2012, [Online; accessed March 2012].

[22] CISCO, "Cisco switched port analyzer (SPAN)," http://www.cisco.com/en/US/products/hw/switches/ps708/products_tech_note09186a008015c612.shtml, 2007, [Online; accessed March 2012].

[23] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix: a tool for generating realistic tcp application workloads in ns-2," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 3, pp. 65–76, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1140086.1140094

[24] K. Vishwanath and A. Vahdat, "Swing: Realistic and responsive network traffic generation," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 3, pp. 712 –725, june 2009.

[25] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1764873.1764876

[26] K. Cho, "WIDE-TRANSIT 150 Megabit Ethernet Trace 2008-03-18 (Anonymized) (collection)," http://imdc.datcat.org, 2012, [Online; accessed March 2012].

[27] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 65–81, Apr. 2003. [Online]. Available: http://doi.acm.org/10.1145/956981.956988

[28] Q. Jia, Z. Wang, and A. Stavrou, "The heisenberg measuring uncertainty in lightweight virtualization testbeds," in *Proceedings of the 2nd conference on Cyber security experimentation and test*, ser. CSET'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855481.1855485

[29] W. J. Conover, *Practical Nonparametric Statistics*. New York: John Wiley & Sons, 1971, pp. 295–301;309–314.

[30] S. Agarwal, J. Sommers, and P. Barford, "Scalable network path emulation," in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, ser. MASCOTS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 219–228. [Online]. Available: http://dx.doi.org/10.1109/MASCOT.2005.61

Christos Siaterlis is an Electrical & Computer Engineer and a project officer at the Joint Research Centre of the European Commission. His research interests include various aspects of the resilience, stability and security of complex systems, and specifically critical infrastructures like the Internet and the Smart Grid. He has a PhD in the area of Internet security management from the National Technical University of Athens and a Master of Science in Computer Science from the University of Southern California, Los Angeles. Before joining the European Commission he was working as a network engineer focusing on network monitoring, traffic measurement and security in WANs.

Andres Perez Garcia is a Telecommunications Engineer, specialty in telematics, from the University of Seville, Spain. He has worked in the private sector (banking and service providers) as a network engineer. Currently he is a network security specialist at the Joint Research Center of the European Commission. His work focuses on inter-domain routing protocols and security of critical networked infrastructures.

Béla Genge received his BSc degree in Computer Science in 2005 from the “Petru Maior” University of Tîrgu Mureş, Romania and his PhD degree in network security in 2009 from the Technical University of Cluj-Napoca, Romania. Currently, he is a Post-Doctoral Researcher at the Institute for the Protection and Security of the Citizen, Joint Research Centre of the European Commission, Ispra, Italy. His research interests include critical infrastructure protection, intrusion detection systems, security and resilience of networked industrial control systems.