Comparing reproduced cyber experimentation studies across different emulation testbeds

Thomas Tarman
Trevor Rollins
Laura Swiler
Jerry Cruz
Eric Vugrin
tdtarma@sandia.gov
Sandia National Laboratories
Albuquerque, New Mexico, USA

Hao Huang
Abhijeet Sahu
Patrick Wlazlo
Ana Goulart
Kate Davis
Texas A&M University
College Station, Texas, USA

ABSTRACT

Cyber testbeds provide an important mechanism for experimentally evaluating cyber security performance. However, as an experimental discipline, reproducible cyber experimentation is essential to assure valid, unbiased results. Even minor differences in setup, configuration, and testbed components can have an impact on the experiments, and thus, reproducibility of results. This paper documents a case study in reproducing an earlier emulation study, with the reproduced emulation experiment conducted by a different research group on a different testbed. We describe lessons learned as a result of this process, both in terms of the reproducibility of the original study and in terms of the different testbed technologies used by both groups. This paper also addresses the question of how to compare results between two groups' experiments, identifying candidate metrics for comparison and quantifying the results in this reproduction study.

CCS CONCEPTS

• **General and reference** → **Experimentation**; **Metrics**; • **Networks** → **Network experimentation**.

KEYWORDS

emulation-based testbeds, cyber experimentation reproducibility, experiment comparison metrics

ACM Reference Format:
Thomas Tarman, Trevor Rollins, Laura Swiler, Jerry Cruz, Eric Vugrin, Hao Huang, Abhijeet Sahu, Patrick Wlazlo, Ana Goulart, and Kate Davis. 2021. Comparing reproduced cyber experimentation studies across different emulation testbeds. In *Proceedings CSET'21.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

CSET'21, August 9, 2021, Virtual Workshop

https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Cyber testbeds have been established over the past two decades to provide a platform for research and experimentation on networks [8]. Some examples include Emulab [28], DETER [6, 19], Common Open Research Emulator (CORE) [3], `minimega` [17], and DARPA's National Cyber Range [10]. Uses of cyber testbeds include test and evaluation, identification of network performance, cyber security investigation, and training [14, 24]. Experimental frameworks such as DEW [18] have been developed to run structured experimental scenarios on these testbeds.

In recent years, there has been increasing interest in reproducibility in cyber experimentation. Reproducible results are foundational in science because they provide evidence that discoveries are completely documented and provide assurance that reported results are not biased. On the one hand, reproducibility in cyber experimentation should be easier than for other scientific disciplines because all of the experimental artifacts (e.g. source code, software for experimental infrastructure, in-experiment applications, configuration files, etc.) can be made available to other research groups to download, install, and run. On the other hand, however, small differences in cyber experimental setups can lead to large changes in results, making reproduction more difficult but also more urgent/necessary. These small differences can include different experimental cluster/testbed host configurations, imprecise synchronization of clocks between cluster nodes and in-experiment virtual machines (VMs) leading to differences in experimental sequencing, different hypervisors used to implement in-experiment VMs, and lack of control over random number seeds used by operating systems and applications. A recent ACM survey on reproducibility in scientific computing [15] describes many challenges similar to those faced in reproducible cyber experimentation, including differences in operating systems, software and hardware, kernels, system resources, etc. A more general discussion of scientific replication and reproducibility is found in [21].

This paper describes a reproduction study of the `minimega` experimental results described in [32] on adversary port discovery versus detection by intrusion detection system (IDS). The original study considered a scenario where an attacker has a malware presence in a power grid supervisory control and data acquisition (SCADA) system. The study developed a mathematical model to describe the time evolution of an attacker's success in discovering vulnerable SCADA devices, and a defender's ability to detect the attacker's

discovery attempts. The mathematical model was validated against experiments using the `minimega` emulation-based testbed. Subsequently, the Sandia National Laboratory (SNL) authors of that paper worked with researchers from Texas A&M University (TAMU) to deploy the experiment on the TAMU CORE emulation-based testbed, collect results, and compare the TAMU results against the `minimega` results obtained in the original experiments. The purpose of these comparisons is to determine the degree to which the emulation-based experiments described in [32] are reproducible on a different testbed, and to measure the degree of correspondence between the original and reproduced experiments.

An experiment is reproducible if results from the same experiment can be independently obtained by different researchers, possibly with different tools [15]. The intent behind this work is to evaluate the reproducibility of [32] and understand the degree of coordination required between research groups to reproduce the paper's results. In addition, the process of reproducing the original experiment led to questions about how experimental results should be compared, which metrics should be used, how closely do they need to agree (given inherent randomness associated with emulation-based experiments), and ultimately, how to consider when a cyber experimental result is indeed reproduced.

This paper is organized as follows. We first introduce the experiment that we considered in this reproduction study. We then describe the testbeds that were used in the original study and in the reproduced study, highlighting the differences between these testbeds. We describe the experimental design and how results were collected, and provide observations regarding the process of reproducing the experiment. We then define different metrics for *quantitatively* comparing these results and apply each of these metrics to both sets of results. Analysis and observations regarding these comparisons are provided next, and the paper concludes with summary remarks and suggestions for future research.

2 ASSESSING DETECTION OF ADVERSARY NETWORK SCANNING IN EMULATION

In this section we describe how we assess detection of adversary network scanning using emulation. First, we describe the topology and scenario considered in this paper, and then describe how it is modeled in the two different emulation testbeds used in this reproducibility study. The processes used for running the scenarios and collecting data are described, and this section concludes with some discussion regarding the amount of interaction required between the two groups to reproduce the experiment.

2.1 Scanning/detection topology and scenario

The network scanning/detection experiments described here are motivated by the CRASHOVERRIDE attack used in the 2016 Ukraine power grid attack [2]. We assume the attacker, which has a presence on the industrial control system (ICS) network, is scanning the network to identify the IP addresses and port numbers associated with vulnerable remote terminal units (RTUs). Concurrently, the defender is using an IDS to detect the attacker's scanning activity. Following the analysis described in [32], we measure the number of vulnerable RTUs discovered with respect to time, and measure the detection probability and the time that the attacker's scans are detected.

The experimental setup, described in detail in [32], includes VMs that represent the Nmap scanning, Snort detection, router, and RTU nodes. The scanning node uses Nmap to identify the IP addresses of RTU nodes with an open port that represents a "vulnerable" service [13]. Nmap performs this discovery in two stages: it uses Internet Control Message Protocol (ICMP) echo/echo-reply messages to discover available hosts, and it uses Transmission Control Protocol (TCP) connection establishment requests to discover open, closed, or filtered (non-responsive) ports. (In the experiment, Nmap is configured to probe only one port per host, to prevent any instabilities that may result from scanning many ports per host.) Nmap provides command-line parameters to configure its operation, e.g. to evade detection by an IDS. For example, the user can use the (*randomize-hosts*) parameter to cause Nmap to search host IP addresses randomly rather than sequentially, to make the scan less obvious to IDS. We found the randomization of port order in the scanning to be a critical issue in reproducibility, as discussed in Section 3, and cannot be exactly replicated in the emulations because `minimega` and CORE do not allow control over random number seeds. Additionally, the host group size can be adjusted using the *max-hostgroup* parameter, which along with *scan-delay*, can be used to adjust the intensity of the scan. Decreasing *max-hostgroup* and increasing *scan-delay* will cause the scan to be less detectable by an IDS, but it will increase the amount of time required to scan all available hosts and ports.

The experimental setup uses Snort as the intrusion detection system, which was selected because it is open source and widely used [7]. In the original and reproduced experiments we configure Snort to use the *sfPortscan* module to detect TCP connection requests. With this module (in the "low" setting), Snort counts the number of rejected connections observed within a 60 second time window. If the number of rejected connection attempts exceeds a threshold (5 for the "low" setting), Snort isues an alert, which includes the source and destination IP addresses of the probe that exceeded the threshold.

The experimental topology, Nmap and Snort software, and emulation environments were used in this reproducibility study to run four different scenarios - two attacker strategies combined with two experimental formulations where all randomness is enabled or removed (intermediate formulations of randomness were assessed, but are not presented in this paper for brevity). The outputs of the experiments are the number of open, closed, and filtered ports found as a function of time and the probability that the attacker is detected before time t.

2.2 Emulation experimentation

Emulation-based experimental testbeds were used by both groups to assess port discovery and detection times because these testbeds provide a flexible, scalable platform for constructing and running experiments. Furthermore, emulation (whether based on virtual machines or containers) allows experimenters to use real software for scanning and intrusion detection, increasing the fidelity of the experiment (because the experiment is not using an abstract representation of the code) and increasing the likelihood of successful

reproduction (because the software versions that were used in the original experiment were identified). Although both groups used the same software, software versions, and experimental designs and methods, each group used different emulation testbed technologies. These technologies are described in more detail below.

2.2.1 minimega. `minimega` [17] is a set of open-source tools for deploying and managing experiments using virtual machines. `minimega` uses QEMU [5], KVM [16], and Open vSwitch [12] to create virtual network topologies. In addition, `minimega` includes mechanisms for launching applications and collecting experimental results. In this set of experiments, `minimega` APIs were used to launch and manage Nmap and Snort.

In the original experiment, the VMs were run on a single physical node, with dual 2.1 GHz Intel ES-2683v4 processors with 16 cores each (32 cores per physical node), 512 GB of RAM, running Ubuntu 18.04. We used our own tool called ScOrch (Scenario Orchestrator) to manage experimental components such as topologies, applications, and experimental parameters (similar to DEW [18]), and facilitate data collection for analysis. The topology remains the same across all scenarios - a single control center subnetwork, single SCADA subnetwork consisting of eight substations and three RTUs per substation (for a total of 24 RTUs), and a router that connects both subnetworks. The scanning workstation (the "compromised", or attacker node) is on the control center network and runs Nmap. The scanned hosts (or "victim" nodes) are on the SCADA network, which has an IDS running Snort [7]. SCADA network devices are configured as open (four devices), closed (eight devices), or filtered (12 devices) using the same techniques as described in 2.2.2. In scenarios that require random packet dropping, the nodes' interfaces are configured with the appropriate drop probability (ten percent in these experiments) using the following command: `iptables -A INPUT -p tcp -m statistic –mode random –probability 0.10 -j DROP`.

2.2.2 RESLab testbed background. TAMU's Resilient Energy Systems Laboratory (RESLab) testbed runs real-time power system simulation and communication network emulation with the integration of industrial control and protective devices [23]. The communication network emulation is using CORE [3] in one Linux virtual machine, using FreeBSD jails running in the VM to emulate network nodes. The CORE VM is configured with 16 CPUs, 32 GB memory, and 200 GB hard disk storage.

The suitability of CORE for emulating smart grid networks was shown in [30], where authors compared works on co-simulation and discovered CORE to be suitable for large-scale simulations and also found applications in the testbed to be easily deployable to embedded devices. Similarly, an Army microgrid communication network has been emulated with CORE [31] to demonstrate cyber intrusions and analyze their impacts.

Within CORE, the TAMU testbed establishes the same one control center with eight substation communication network as used in the original study, which is shown in Figure 1. There is one node in the control center subnet that monitors and controls the devices in substations. In each substation, there are three nodes to model three RTUs. The emulated network in CORE has two broadcast subnets: one for the control center, and the other is for the eight substations, with three RTUs per substation. Although the IP addressing scheme in the CORE topology, is different from the `minimega` topology, in both cases the IP addresses are allocated such that Nmap only scans these IP addresses, and doesn't scan others (which could lead to biases in timing results).

Following [32], the same parameters in the RTUs in CORE are set as open, closed, and filtered. The experiment uses *SSH* as a proxy for "vulnerable" ports, with open and closed nodes configured through each host's *SSH* settings, and the filtered node configured using `iptables -A INPUT -p tcp –destination-port 22 -j DROP`. The experiment uses four open, eight closed, and 12 filtered nodes, as in [32]. We also add packet loss possibilities for communication links to emulate different communication scenarios, such as considering ten percent packet drop for each link. In CORE, each node is running as a Linux container (using FreeBSD jails), unlike in `minimega` where the nodes are each hosted in a VM (KVM).

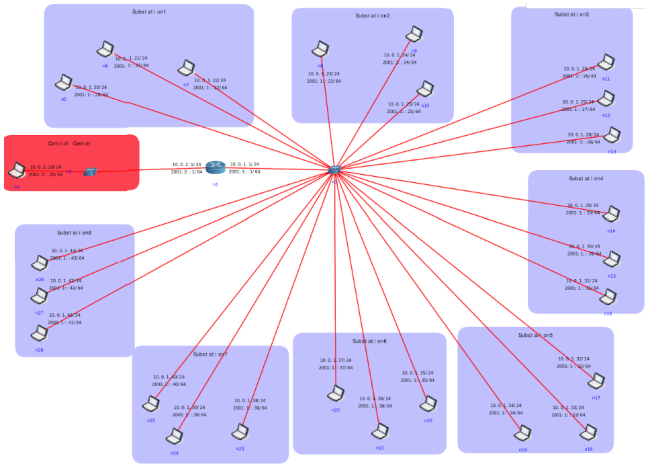


Figure 1: Eight-Substation Communication Network Emulation in CORE

2.2.3 Testbed comparison. In comparing the two testbeds, one question is whether differences in experimental results arise due to the use of containers in RESLab versus VMs in `minimega`. A primary difference between VMs and containers is that containers provide a way to virtualize an operating system (OS) so that multiple workloads can run on a single OS instance. Unlike containers, in VMs, the hardware is being virtualized to run multiple OS instances managed through a hypervisor. Containers, having limited resource access, face challenges hosting resource-intensive applications like Snort. Still, the advantage of using a container over a VM is that the application or service running in a VM is OS dependent, but containers sharing a common OS provide a convenient platform to run multiple executables. Hence, containers are preferred when the priority is to maximize the deployment of a number of applications on a minimal number of servers [9]. Containers are usually light-weight [27], with a minimal startup time [25], as well as requiring less memory space, making them a viable option for large scale deployment. From the backup and restoration aspect, containers are less reliable, as a single glitch in the OS can affect all the containers whereas VMs are fully isolated, so they are more

Sources of Randomness	Deterministic Formulation	Stochastic Formulation
nmap address search order	fixed sequence same for each trial	random sequence varies between trials
packet loss	no loss (none)	random drops with prob=0.1

Table 1: Experimental Design Controlling for the Sources of Randomness

reliable. Additionally, VMs enable diversity in running OS-specific applications. Container-based testbeds face more challenges when scenarios require execution of different OS environments.

2.3 Reproducing the results - experiment design and data collection

The experimental design was the same for `minimega` and CORE. The experiments used two formulations of randomness in the experiment, summarized in Table 1. The first formulation tested involved no randomness: the node ordering for the Nmap address search was fixed, and there were no dropped packets. The second formulation used random port ordering and dropped packets. Intermediate formulations (e.g. fixed Nmap node ordering, with dropped packets) were also evaluated, but results are not presented here for brevity.

We ran 100 replicates of the formulations listed in Table 1 for each of two attacker strategies. The first attacker strategy consists of a "slow and stealthy" attacker. This corresponds to a host group size of four and a delay of ten seconds. The second attacker strategy consists of a "fast and loud" attacker, which corresponds to a host group size of six and a delay of five seconds. We ran the four experimental combinations of randomness in the port ordering and packet drops listed in Table 1 for each of these two strategies. This involved eight experimental scenarios, where each experimental scenario involved 100 replicates.

To efficiently reproduce the experiment in RESLab, a multi-thread Python script is used to run the Nmap port scan and Snort intrusion detection within CORE to collect experiment data in the same way as in the `minimega` experiments, as shown in Algorithm 1. After running the configured network in CORE, the Python script will start Snort and run the Nmap port scan n times to collect the Nmap port scan file and Snort alert and log files separately for each experiment. Additionally, we store the traffic data in Wireshark pcap files as a reference. The Snort IDS runs in the backbone network in the VM, and Nmap runs in the emulated network in CORE. If the Snort keeps running, the Snort alert and log files will save all Nmap port scan experiments, which makes it hard to synchronize the results for each experiment. Due to limited resources with containers, we have to kill the process of Snort with `kill -9 'pgrep snort'` for each experiment as shown in Algorithm 1. For each Nmap port scan experiment in RESLab, the script starts the Wireshark instance in CORE to capture all traffic in the network. Then, the script starts both Snort for intrusion detection and Nmap for port scanning and applies the strategies specified in Table 1. Once Nmap is finished, the script saves all results from Wireshark, Snort, and Nmap into specific files. After the files have been saved, the script stops Snort until the next experiment starts.

The inputs are the Nmap scanning parameters, the number of experiments n and the commands to start Wireshark, Nmap and Snort. The value of n determines how many experiments will be performed for the specified Nmap port scan strategy with the current network configuration. With the *host-group* and *scan-delay* parameters, the Nmap command specifies the Nmap strategy to be either *Slow and Stealthy* or *Fast and Loud*. The *scan-order* parameter determines how Nmap scans ports in the network, *Random* or *Ordered*. The *Random* scan sequence uses the *randomize* command, while the *Ordered* scan sequence uses `-iL` with a predefined scan sequence file. After each experiment run is completed, results are saved in the Wireshark pcap files, Nmap port scan files, and Snort alert and log files.

Algorithm 1 Configuration Emulation Environments

1: Input = Nmap scanning parameters: *max-host-group min-host-group scan-delay*, *scan-order*, number of experiments: n
2: **for** i in (1, n) **do**
3: Start Wireshark and Snort
4: Perform Nmap port scan
5: Save Wireshark, Snort, Nmap files to Folder i
6: Stop Snort
7: Stop Wireshark
8: **end for**
9: Output = wireshark pcap files, Nmap port scan files, Snort alert and log files

2.4 Reproducibility observations

The purpose of this work was to assess the reproducibility of the experiments described in [32]. Ideally, all of the information needed to reproduce the experiment would be contained in the original paper and in supplemental sources, e.g. experiment artifact repositories. However, our experience with this study showed that there was still a degree of coordination required to reproduce the experiment. Even after sharing experimental artifacts (e.g. `minimega` topology files, ScOrch orchestration files, Snort configuration files, etc.), the teams found that some details were missing. First, the teams had to coordinate on the assignment of open, closed, and fixed ports to the hosts, and the order in which Nmap would scan them. The original paper did not consider a fixed order scan, so the ordering detail was not included.

Additionally, though the original paper described that `iptables` were used to simulate dropping of packets, the specific details of the `iptables` configurations were not included. Hence, the teams discussed `iptables -A INPUT -p tcp –destination-port 22 -j DROP` for filtered nodes and `iptables -A INPUT -p tcp -m statistic –mode random –probability 0.1 -j DROP` for all nodes with packet loss.

A further issue that was identified was killing Snort between experiment trials. If Snort is not killed between trials, TCP reset packets from a trial can potentially affect the timing of alerts in a subsequent trial. The teams initially noted significant discrepancies between the SNL and TAMU results. It was only after alert and

packet capture files were shared and examined that the teams recognized that one set of experiments killed Snort between trials and the other set did not. Once this issue was recognized and the process of killing Snort between experimental trials was made consistent, results between the two platforms matched up much more closely.

With data successfully collected from both groups' testbeds, processing and data analysis of experimental results from both testbeds was performed by one of the research groups, and is described in the next section.

3 COMPARING RESULTS FROM TWO EMULATION TESTBEDS

Each replicate of the eight experimental scenarios outlined above (fast/slow, packet drop/no drop, random/deterministic host order) produced a time series of ports discovered over time. For example, at a given time, say 30 seconds, there may be two open ports discovered, five closed ports discovered, and nine filtered ports discovered. Since there are 100 instances of these time series for each experimental scenario, we can calculate statistics at each time point such as the mean number of closed ports discovered at each time point or the variance of the closed ports discovered at each time point. Furthermore, the 100 points can be used to generate a cumulative distribution function (CDF) of ports discovered at each time point. The cumulative distribution function is the probability that a random variable (e.g., number of closed ports found) is less than x, where x ranges from zero to eight closed ports in these particular studies. The CDF is a probability distribution, so its values are bounded between zero and one.

An example of the CDF plot and its construction from the 100 replicates at a particular time point is shown in Figure 2. Because we have distribution information in the form of a CDF at each time point, we can use statistical metrics for comparing distributions. These are outlined in Section 3.2 below. First, we make some general observations about metrics in the next section.

3.1 Definition of comparison metrics

Reproducibility involves comparison between two sets of results, and using metrics to measure the degree of similarity between the experiments. In this paper, this involves comparing results from two cyber emulation platforms, `minimega` and CORE. As we investigated metrics with which to compare the experimental results outlined above (e.g. replicates of time series data), we were guided by principles from the validation community. There is a growing community focused on the validation of computational model results (typically, these are physics or engineering models involving the solution of partial differential equations) [22]. In the context of computational simulation, *validation* refers to the comparison of a computational simulation with physical experiments *to determine if the estimated accuracy of the model is adequate for the intended use of the model* [22]. Many validation metrics have been defined to help make the comparison, but the degree of mismatch accepted (e.g. accuracy requirement) between the model and physical experiment is typically defined by the decision maker or ultimate user of the model.

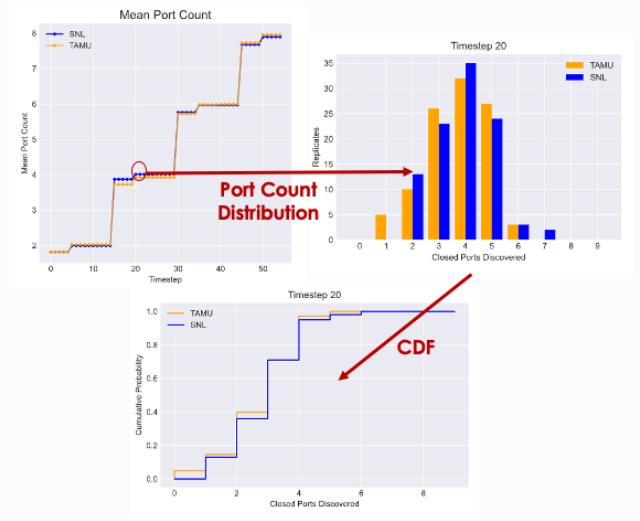


Figure 2: A CDF of port discovery is constucted from the 100 replicates at each timestep. Shown is the "closed" port discoveries at timestep 20 for both SNL and TAMU.

An ideal feature of a validation metric is the inclusion of uncertainties both from the simulation results and the physical experiment results in the metric. As we have stated, the analysis we provide here is not formally validation in the strict sense: we have not run the scanning/detection experiment on physical hardware to compare with the emulation results. However, we can use these principles found in validation for the comparison of emulation results across two or more different emulation platforms.

With that in mind, we identified four metrics to consider for comparing the datasets from the scanning/detection experiments. These metrics are generalizable to any experimental situation where one has replicates from both of the experiment platforms (`minimega` and TAMU CORE in this case). The replicates (e.g. 100 replicates of `minimega` and 100 replicates of CORE) represent inherent variability induced by timing differences, kernels, operating systems, etc. as previously discussed. The metrics compare the distribution from each of these platforms. The metrics are outlined in the section below.

3.2 Metrics considered

The scanning/detection experiments produce time series data at uniform, one-second intervals. Our focus was primarily on understanding the differences in the number of ports discovered over time, which is a discrete variable at each time point. However, we also investigated the differences in alert times which is a continuous variable. The 100 replicates from both TAMU and SNL indicate significant variability in the experiments where random host ordering and/or packet dropping occurs. This variability can be from random ordering of ports in the scanning phases, dropped packets, or differences in the emulation environment. Our focus was to tease out which differences can be attributed to the emulation environment vs. the randomness in aspects of the experiment.

The metrics we investigated are:

3.2.1 t-test. The t-test is a widely used statistical test to determine if there is significant difference between the means of two groups. [20] The test statistic (t) converges to the t-distribution, and allows rejection of the null hypothesis of $\bar{X}_1 = \bar{X}_2$ at various confidence levels. This can be seen in Equation 1. We note that a p-value of 1.0 indicates a perfect match between the means.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_p \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$
$$\textit{where} \tag{1}$$
$$s_p = \sqrt{\frac{(n_1 - 1)\, s_{X_1}^2 + (n_2 - 1)\, s_{X_2}^2}{n_1 + n_2 - 2}}$$

3.2.2 Kolmogorov-Smirnov Test. The Kolmogorov-Smirnov (KS) test is a well known non-parametric statistical test for equality of distributions. [1] The KS distance between two random variables is the maximum vertical distance between their cumulative distribution functions, $CDF_1(x)$ and $CDF_2(x)$, as shown in Equation 2. Note that $CDF_{i,j}(x)$ refers to the CDF from distribution i based on j sample points. The test statistic (D) converges to the Kolmogorov distribution, and allows rejection of the null hypothesis of $CDF_1(x) = CDF_2(x)$ at various confidence levels. Again, a p-value of 1.0 indicates perfect agreement between the distributions.

$$D_{n,m} = \sup_x |CDF_{1,n}(x) - CDF_{2,m}(x)| \tag{2}$$

3.2.3 Area metric. The area metric also quantifies the difference between sample CDFs, however it accounts for the entire difference between the functions rather than just the maximum vertical distance. [11] The metric calculation is shown in Equation 3. Unlike the other tests, the units for this metric are the same as the measurements themselves, and there isn't a formal acceptance measure or statistical test, but a value of 0.0 indicates perfect agreement.

$$A_{n,m} = \Delta(x) \sum_{x=1}^{x_{end}} |CDF_{1,n}(x) - CDF_{2,m}(x)| \tag{3}$$

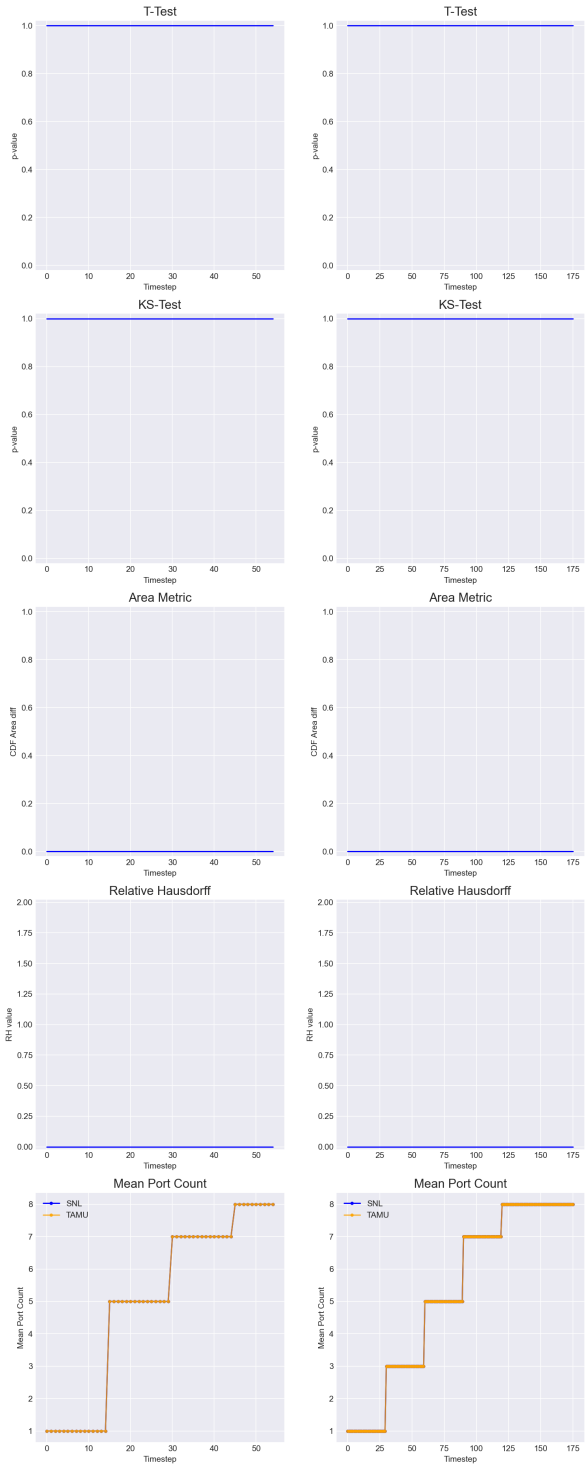
3.2.4 Relative Hausdorff Distance. The Relative Hausdorff distance was originally developed for graph analysis, to compare quantities like the complementary cumulative degree distribution of large graphs [4, 29] The distributions $F_1(x)$ and $F_2(x)$ are (ϵ, δ) close by the Relative Hausdorff distance if they meet the constraints shown in Equation 4.

$$\forall x, \exists x' \in [(1-\epsilon)x, (1-\epsilon)x']$$
$$\textit{such that} \tag{4}$$
$$|F_1(x) - F_2(x')| \leq \delta F_1(x)$$

For our calculations, we used the additional constraint of $\epsilon = \delta$ which weighs the effects of the vertical and horizontal differences equally.

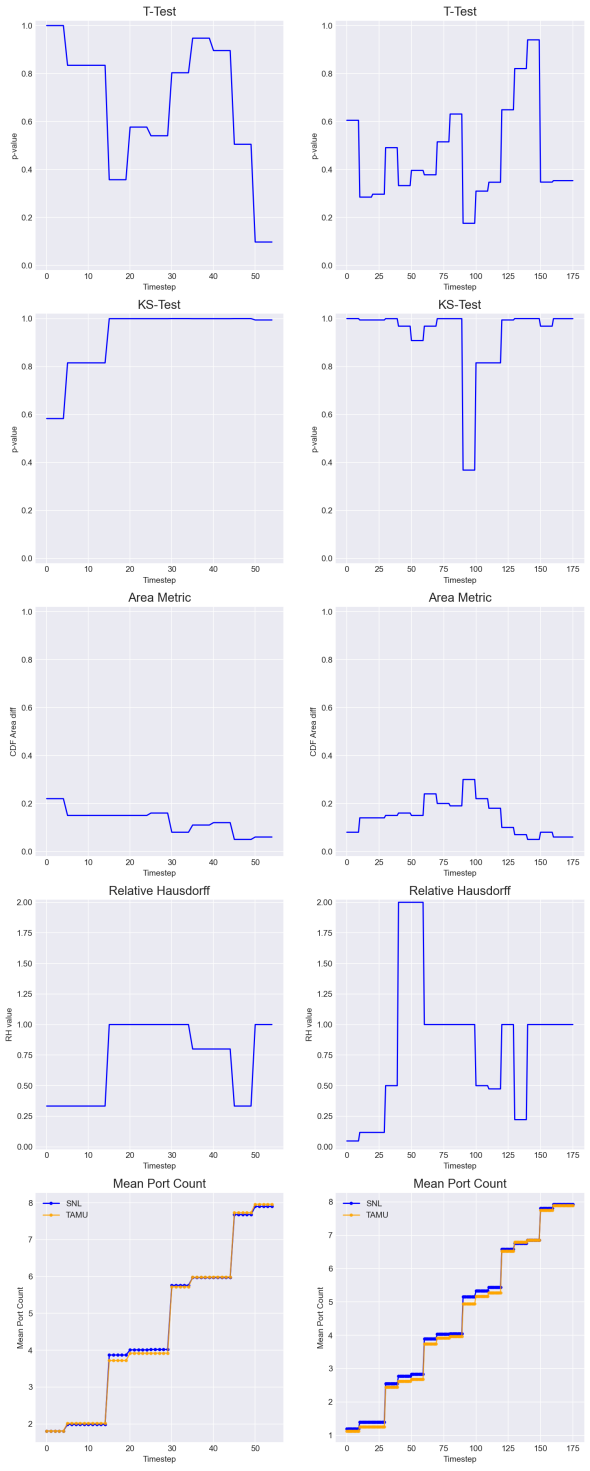
3.3 Analysis and observations

To quantify the experimental differences, our primary focus was on the number of ports discovered over time. We only show results for closed port discovery; the results are similar for open and filtered ports. For each of the eight experimental scenarios described in Section 2.3, this port discovery time series data was compared using



(a) Fast **(b) Slow**

Figure 3: Comparison of closed port discovery for (a) Fast and (b) Slow experiments, deterministic formulation.



(a) Fast (b) Slow

Figure 4: Comparison of closed port discovery for (a) Fast and (b) Slow experiments, stochastic formulation.

the metrics listed above. This in turn results in a time series of metric values comparing the TAMU and SNL runs for each time step for the duration of the experiment and highlights points of variability. Figures 3 and 4 show this comparison for each of the metrics. These figures encompass four of the eight scenarios. Figure 3 shows the closed port discovery for the fully controlled scenario with fixed port ordering and no dropped packets both for the fast and slow attacker strategies. These metrics indicate no variability across the entirety of this fully controlled scenario, which serves as a good baseline to indicate that the two emulation testbeds behave identically in the absence of randomness.

The pair of experiments with both modes of randomness shown in Figure 4 appear to produce similar but not identical mean results. Here, the statistical metrics indicate greater variability between the TAMU and SNL runs. This variability is also visible to the eye, as the plots with the mean port count don't align as well as they do in Figure 3. It is also clear that the fast attacker strategy lends itself to more consistent results than the slow. This makes sense as duration of the experiment is shorter, providing less opportunity for the replicates to become offset in terms of port discovery.

All of the described metrics clearly show there is a difference in variability between the controlled and random experiments. The question remains as to which metrics are most useful for comparing the variability within an experiment. Ultimately we find that the combination of the KS-test and area metrics is most suited to this task.

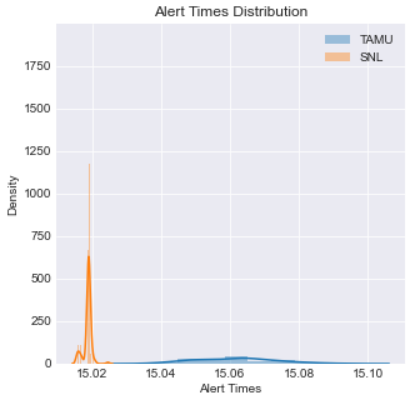
We find the t-test is a poor metric for this discrete data, as this test is looking at the difference in means rather than the distribution. Thus a single step up in port count for a number of the replicates (e.g. 3 found ports to 4), can result in a very different mean to the other group as well as a larger variance. This can be seen when looking at Figure 4 - the resulting t-test p-values show far more variability than those of the KS-test.

Additionally, we find the Relative Hausdorff distance doesn't completely suit our use-case as it tends not show the nuance of the variability well. It instead has a tendency to make large jumps in magnitude, often to discrete values. This is likely due to the fact that though it was designed with discrete data in mind, the CDFs that we are evaluating are much narrower in range, and far more stair-stepped in nature. This seems to manifest itself as these curves having a minimum Relative Hausdorff distance of 1.0 as soon as they become offset.

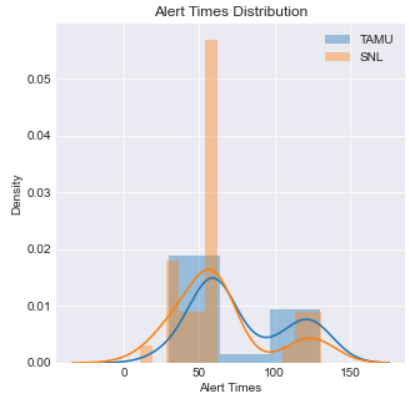
The area metric, in contrast, highlights small fluctuations values that correspond to the differences between the CDFs. This metric provides us with a better tool for teasing out small differences between the environments. However there is no formal acceptance metric or statistical test for this value. This then leaves us with designating an arbitrary threshold for what is deemed "similar."

The KS-test provides more clarity in this regard. The test statistic converges to the Kolmogorov distribution, and allows rejection of the null hypothesis of $CDF_1 = CDF_2$ at various confidence levels. If the p-value is greater than 0.05 (at a 95% confidence level), we cannot reject the null hypothesis that the distributions are the same, so the CDFs would be considered similar with larger p-values indicating more similarity. These p-values often mirror the fluctuations seen in the area metric, but are not always quite as nuanced. Many small differences in the CDFs are missed, as the p-value remains

1.0 during these fluctuations. Hence, when looked at together, this p-value and area metric provide a nice mix of statistical significance and insight into the slight differences between experiments.



(a) Fast discovery, deterministic formulation



(b) Slow discovery, stochastic formulation

Figure 5: Alert time distributions of (a) the least and (b) the most variable experimental scenarios when comparing port discovery.

Though port discovery was the main focus of our comparison, we also looked at the distribution of alert times for the 100 replicates of each run. This data has the advantage of being neither a time series, nor discrete values which seems to lend itself well to comparison by the KS-test. However in reality the results can sometimes contradict what is seen when comparing the port discovery data.

Figure 5 shows the alert time distribution for two of the experimental scenarios. Figure 5a displays the distributions of alert times for the least variable scenario when comparing port discovery. However as can be seen, the alert time distributions appear quite different. This is born out by a p-value of 0 when comparing these curves with the KS-test. In reality we can see that the raw alert times of these experiments differ on the millisecond scale, but because they are so tightly clustered, their distributions are significantly offset.

Figure 5b shows the distributions of alert times for the most variable scenario when comparing port discovery. Here the opposite phenomenon is at play. Alert times in this scenario differ on the scale of tens of seconds, but because they are more dispersed, the distributions actually appear quite similar and yield a KS-test p-value of 0.155.

This illustrates that strictly comparing the alert time distributions of two sets of experiments may not be sufficient to quantify the variability between them. If we want alert times to fall into the same distribution, the experiments in Figure 5a would not be considered the same. However if we want the alert times to only differ by some small margin, these experiments would indeed be considered similar. Comparison metrics depend on the question being asked. It may not matter if the statistical tests indicate the distributions are different across platforms but the differences are so small that for practical purposes, we would consider them the same. When comparing emulation results across platforms, one must carefully consider the question being addressed and identify thresholds for absolute differences in mean results, as well as full statistical comparisons.

4 CONCLUSIONS AND FUTURE WORK

Reproducible results are essential for progress in cyber experimentation, and this paper describes a case study in reproducing the cyber experimental results described in [32]. Ideally the originally published study, along with any experimental artifacts publicly located in a repository, would be sufficient for a second research group to reproduce the original study. In this case, differences in testbed technologies and missing details encountered while reproducing the experiment required some coordination between the groups before the second group started to produce similar results. Further research and capabilities, such as those being developed by the Sharing Expertise and Artifacts for Reuse through Cybersecurity Community Hub (SEARCCH) project [26], are needed for comprehensively sharing experimental artifacts. In addition, further research is needed for understanding differences between common cyber experimentation platforms, so those differences can be considered when comparing experimental results. This paper also identified a need to consider different metrics for quantitatively comparing results from the two sets of experiments, depending (for example) on whether an analyst wants to observe differences in average results across testbed environments, and if an analyst wants to use an accepted statistical test to determine if the distribution of results are similar. Additional research in applying other existing metrics or developing new ones is needed, along with additional guidance on when to use these metrics to compare experiments for reproducibility or validation studies. Additional reproducibility studies on other cyber experiments are needed to test the generalizability of these metrics.

ACKNOWLEDGMENTS

This work has been supported by the LDRD Program at the Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology &

Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] 2012. *NIST/SEMATECH e-Handbook of Statistical Methods*. Technical Report. National Institute of Standards. https://doi.org/10.18434/M32189

[2] 2017. *CRASHOVERRIDE Analysis of the Threat to Electric Grid Operations*. Technical Report version 2.20170613. Dragos Inc., Hanover, MD.

[3] Jeff Ahrenholz, Claudiu Danilov, Thomas R Henderson, and Jae H Kim. 2008. CORE: A real-time network emulator. In *MILCOM 2008-2008 IEEE Military Communications Conference*. IEEE, 1–7.

[4] Sinan G Aksoy, Kathleen E Nowak, Emilie Purvine, and Stephen J Young. 2019. Relative Hausdorff distance for network analysis. *Applied Network Science* 4, 1 (2019), 80.

[5] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX Annual Technical Conference, FREENIX Track*, Vol. 41. 46.

[6] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, and John Wroclawski. 2009. Current developments in DETER cybersecurity testbed technology. In *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE, 57–70.

[7] Cisco. 2019. Snort intrusion detection and prevention system. https://www.snort.org/.

[8] Jon Davis and Shane Magrath. 2013. *A survey of cyber ranges and testbeds*. Technical Report. Cyber and Electronic Warfare Division, Defence Science and Technology Organization, Australian Government.

[9] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. 2015. An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 171–172. https://doi.org/10.1109/ISPASS.2015.7095802

[10] Bernard Ferguson, Anne Tall, and Denise Olsen. 2014. National cyber range overview. In *2014 IEEE Military Communications Conference*. IEEE, 123–128.

[11] Scott Ferson, William L Oberkampf, and Lev Ginzburg. 2008. Model validation and predictive capability for the thermal challenge problem. *Computer Methods in Applied Mechanics and Engineering* 197, 29-32 (2008), 2408–2430.

[12] Linux Foundation. 2019. Open vSwitch. http://www.openvswitch.org/

[13] G. Lyon. 2019. Nmap: the network mapper. http://nmap.org/.

[14] Alefiya Hussain, David DeAngelis, Erik Kline, and Stephen Schwab. 2020. Replicated Testbed Experiments for the Evaluation of a Wide-range of DDoS Defenses. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 46–55.

[15] Peter Ivie and Douglas Thain. 2018. Reproducibility in scientific computing. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–36.

[16] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux symposium*, Vol. 1. Dttawa, Dntorio, Canada, 225–230.

[17] minimega developers. 2019. minimega: a distributed VM management tool. http://minimega.org/

[18] Jelena Mirkovic, Genevieve Bartlett, and Jim Blythe. 2018. DEW: Distributed Experiment Workflows. In *11th USENIX Workshop on Cyber Security Experimentation and Test CSET 18)*.

[19] Jelena Mirkovic, Terry V Benzel, Ted Faber, Robert Braden, John T Wroclawski, and Stephen Schwab. 2010. The DETER project: Advancing the science of cyber security experimentation and test. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 1–7.

[20] Douglas C Montgomery and George C Runger. 2010. *Applied statistics and probability for engineers*. John Wiley & Sons.

[21] Engineering National Academies of Sciences and Medicine. 2019. *Reproducibility and Replicability in Science*. The National Academies Press, Washington, DC. https://doi.org/10.17226/25303

[22] William L Oberkampf and Christopher J Roy. 2010. *Verification and validation in scientific computing*. Cambridge University Press.

[23] Abhijeet Sahu, Patrick Wlazlo, Zeyu Mao, Hao Huang, Ana Goulart, Katherine Davis, and Saman Zonouz. 2020. Design and Evaluation of A Cyber-Physical Resilient Power System Testbed. *arXiv preprint arXiv:2011.13552* (2020).

[24] Stephen Schwab and Erik Kline. 2019. Cybersecurity Experimentation at Program Scale: Guidelines and Principles for Future Testbeds. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 94–102.

[25] Kyoung-Taek Seo, Hyun-Seo Huang, Young Moon, Oh-Toung Kwon, and Byeong-Jun Kim. 2014. Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters* (2014). http://www.mmc.igeofcu.unam.mx/acl/femp/MaquinasVirtuales/CursoMaquinasVirtuales/VirtualizacionEnLinuxCon-HPC-VPS/25.pdf

[26] Sharing Expertise and Artifacts for Reuse through Cybersecurity Community Hub (SEARCCH) project. 2021. https://searcch.cyberexperimentation.org/.

[27] S. Shirinbab, L. Lundberg, and E. Casalicchio. 2017. Performance evaluation of container and virtual machine running cassandra workload. In *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*. 1–8. https://doi.org/10.1109/CloudTech.2017.8284700

[28] Christos Siaterlis, Andres Perez Garcia, and Béla Genge. 2012. On the use of Emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys & Tutorials* 15, 2 (2012), 929–942.

[29] Olivia Simpson, C Seshadhri, and Andrew McGregor. 2015. Catching the head, tail, and everything in between: A streaming algorithm for the degree distribution. In *2015 IEEE International Conference on Data Mining*. IEEE, 979–984.

[30] S. Tan, W. Song, Qifen Dong, and L. Tong. 2012. SCORE: Smart-Grid common open research emulator. In *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*. 282–287. https://doi.org/10.1109/SmartGridComm.2012.6485997

[31] V. Venkataramanan, A. Srivastava, and A. Hahn. 2016. Real-time co-simulation testbed for microgrid cyber-physical analysis. In *2016 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. 1–6. https://doi.org/10.1109/MSCPES.2016.7480220

[32] Eric Vugrin, Jerry Cruz, Christian Reedy, Thomas Tarman, and Ali Pinar. 2020. Cyber threat modeling and validation: port scanning and detection. In *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*. Association for Computing Machinery.