SANDIA REPORT

SAND2019-12785
Unlimited Release
Printed October 2019

Quantifying Uncertainty in Emulations: LDRD Report

Jonathan Crussell, Aaron Brown, Jeremy Kyle Jennings, David Kavaler, Thomas M. Kroeger, Cynthia A. Phillips

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2019-12785
Unlimited Release
Printed October 2019

Quantifying Uncertainty in Emulations: LDRD Report

Jonathan Crussell[1], Data Science & Cyber Analytics
Sandia National Laboratories
Mail Stop 9158
PO Box 969
Livermore, CA 94551-0969

Aaron Brown[2], Cyber Systems Assessments
Sandia National Laboratories
Mail Stop 9158
PO Box 969
Livermore, CA 94551-0969

Jeremy Kyle Jennings, Data Science & Cyber Analytics
Sandia National Laboratories
Mail Stop 9158
PO Box 969
Livermore, CA 94551-0969

David Kavaler, Data Science & Cyber Analytics
Sandia National Laboratories
Mail Stop 9158
PO Box 969
Livermore, CA 94551-0969

Thomas M. Kroeger, Enterprise Cyber Security
Sandia National Laboratories
Mail Stop 9158
PO Box 969
Livermore, CA 94551-0969

Cynthia A. Phillips, Computing Research
Sandia National Laboratories
Mail Stop 1326
P.O. Box 5800
Albuquerque, NM 87185-1326

Abstract

This report summarizes the work performed under the project "Quantifying Uncertainty in Emulations." Emulation can be used to model real-world systems, typically using virtualization to run the real software on virtualized hardware. Emulations are increasingly used to answer mission-oriented questions, but how well they represent the real-world systems is still an open area of research. The goal of the project was to quantify where and how emulations differ from the real world. To do so, we ran a representative workload on both, and collected and compared metrics to identify differences. We aimed to capture behaviors, rather than performance, differences as the latter is more well-understood in the literature.

This report summarizes the project's major accomplishments, with the background to understand these accomplishments. It gathers the abstracts and references for the refereed publications that have appeared as part of this work. We then archive partial work not yet ready for publication.

[1]Principal Investigator

[2]Remaining authors ordered alphabetically by last name

Acknowledgment

This work was funded by the Sandia National Laboratories' Laboratory Directed Research and Development (LDRD) program in the Energy and Homeland Security investment area.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and List of Accomplishments

This work summarizes the results of a 3-year Laboratory Directed Research and Development project that ended in September 2019. Our goals were to:

- Understand the accuracy and limitations of Emulytics
- Develop a knowledge base of how Emulytics differs from the real world
- Provide rules of thumb to Emulytics experimenters

1.1 Emulytics

Emulytics is a Sandia National Laboratories program focused on scientifically rigorous study of the behavior of complex, distributed cyber systems [1]. Emulytics combines large-scale emulation, modeling, and analysis tools to support predictive modeling, training, testing and evaluation, and resilient system design. Through emulation, Emulytics can repeatedly and cheaply create elaborate and intricate testbeds representing systems such as enterprises, internet service providers, and control systems.

Unlike simulations, emulations run the real software on virtualized hardware. Simulations can be used to understand complex systems but all details and behaviors of those systems need to be implemented in the simulator. For complex, distributed systems, this can be very challenging. Virtualized hardware allows us to run the real operating systems and applications, improving the fidelity of the model. However, this hardware abstraction may introduce artifacts that affects experiments.

Given Sandia's increasing reliance on Emulytics to answer mission-oriented questions, it is prudent that we develop a better understanding of the accuracy, and limitations, of Emulytics. To date, the Emulytics community has been heavily engaged in answering questions and developing capabilities and has had little time to study the limitations and accuracy of the emulations themselves. Instead, they have been forced to rely on empirical data and their in-depth knowledge of operating systems, networks, and virtualization to help partners develop questions that they believe Emulytics can answer. This dulls our capability in at least two ways: 1) we may not fully anticipate how the models are wrong, causing misleading results, and 2) we may undersell the capability causing partners to seek alternative, likely more expensive, solutions.

In the following discussion, *host* refers to hardware running the emulation. For example, the host operating system is the one installed on the machine directly, interacting with the hardware. The *guest* system is the one running in the virtual environment.

1.2 Approach

Our goals were to develop a set of principles and procedures to verify and quantify the usefulness of our imperfect Emulytics models by comparing measurements collected from real systems to those collected from their emulations. The core measurement, verification and quantification work here is fundamental research, critical to our Emulytics capability. While the knowledge gained will provide insights that are applicable across a wide range of missions and customers, this research is outside the scope of their mission-driven charters. Nevertheless these insights provide great value across a broad range of national security challenges.

To ensure we base our verification efforts on meaningful metrics, we conducted a survey of the Emulytics community on past and current experiments. Specifically, we identified the questions being asked, the recorded metrics, and how these metrics answered those questions across the Emulytics portfolio. This survey identified the properties of real systems that the emulations must capture for current and future Emulytics experiments to produce useful results. To date, Emulytics has been applied in numerous testing and evaluation experiments which provide natural metrics for comparison such as whether a device under test performs at the desired bandwidth. We drew from Stevens's typology [13] to broaden these metrics to capture different scales of measure. For example, a device's ability to perform could be categorized on a nominal scale as LIKELY, MAYBE, or NOT LIKELY. Other scales include ordering, interval (e.g. temperature), and ratio (e.g. duration). We then devised experiments to test multiple types of metrics on both real systems and their emulations to quantify the magnitude and nature of their differences.

We focused on experiments on isolated and controlled networks where the fidelity of the host is preserved. Our experiments involved a simple workload: an HTTP client requesting web pages from an HTTP server. We explore two variants of this workload: loading small, 500B pages and large, 16MB images (we refer to these as the *http500B* and *http16MB* workloads, respectively). We selected this workload because HTTP is pervasive in computing systems and but is a relatively simple application protocol so we can understand it. HTTP relies on TCP, a pervasive, connection-oriented protocol that is used by many other applications as well for reliable communication. Therefore, by using HTTP as our workload, we can learn about how other applications may behave as well. Specifically, for TCP, we are interested in comparing properties of the data stream such as the number of in-flight packets, the retransmission rate, and how congestion control performs. This knowledge can help us to better understand how applications that use TCP may differ in our Emulytics models.

During these experiments, we aimed to understand how adjusting parameters of the emulation affects the model usefulness. These parameters are numerous: the virtualization technology, the host operating system, the host kernel scheduler, the guest operating systems, the virtual network interface drivers, the bandwidth of the emulated network, the oversubscription rate, the virtual machine workload, and the underlying hardware. Testing all of these was infeasible so we instead used the survey to select parameters that are more likely to be changed during Emulytics experiments.

We identified many more tests and parameters that we would like to investigate and are exploring follow-on funding to do so. In Section 4, we list these ideas for future work.

1.3 Toolset

We heavily rely on the open-source *minimega* toolset [9] to support our research. This toolset has been developed by Sandia over nearly a decade as part of its Emulytics program. Traditionally, the toolset has been focused toward instantiating large networks of virtual machines once or a small number of times. For example, in early work [10] in the precursor to the *minimega* toolset, the goal was to boot one million virtual machines (VMs).

1.4 Accomplishments

In this section, we document a high-level overview of our accomplishments. We:

- Conducted a survey of the Emulytics portfolio which informed our work and has been internally useful for knowledge sharing amongst the Emulytics community.
- Created a methodology to compare virtual and physical testbeds that can be reused as the testbeds evolve. This methodology is captured in a peer-review publication.
- Developed techniques to compare sequences of system calls to measure differences in how applications interact with the operating systems.
- Discovered and documented notable differences between virtual and physical testbeds in the underlying packetization.
- Developed a deeper understanding of how oversubscription affects virtual testbeds.
- Captured lessons learned in building, running, and analyzing data from our experiments in a peer-reviewed workshop publication.

- Supported several projects indirectly through the tools and processes developed by this LDRD. Most notably, this led to a peer-reviewed publication with the SECURE Grand Challenge [12].
- Identified and documented many experiments to run in future work to further quantify where and how Emulytics models are useful.
- Improved the underlying toolset to facilitate running thousands of experiments. Specifically, our contributions are included in three *minimega* releases: v2.4, v2.5, and v2.6.

Chapter 2

Summary of Published Papers

In this section, we list the two published papers that were directly supported by this LDRD and one published paper that was indirectly supported by this LDRD. We omit the details of these published works as they have already been archived elsewhere. The remainder of this report focuses on archiving experiments, results, and ideas that are not yet archived elsewhere.

Crussell, Jonathan, Thomas M. Kroeger, Aaron Brown, and Cynthia Phillips. "Virtually the Same: Comparing Physical and Virtual Testbeds." In 2019 International Conference on Computing, Networking and Communications (ICNC), pp. 847-853. IEEE, 2019 [5].

Abstract: Network designers, planners, and security professionals increasingly rely on large-scale testbeds based on virtualization to emulate networks and make decisions about real-world deployments. However, there has been limited research on how well these virtual testbeds match their physical counterparts. Specifically, does the virtualization that these testbeds depend on actually capture real-world behaviors sufficiently well to support decisions?

As a first step, we perform simple experiments on both physical and virtual testbeds to begin to understand where and how the testbeds differ. We set up a web service on one host and run ApacheBench against this service from a different host, instrumenting each system during these tests. We define an initial repeatable methodology (algorithm) to quantitatively compare physical and virtual testbeds. Specifically we compare the testbeds at three levels of abstraction: application, operating system (OS) and network. For the application level, we use the ApacheBench results. For OS behavior, we compare patterns of system call orderings using Markov chains. This provides a unique visual representation of the workload and OS behavior in our testbeds. We also drill down into read-system-call behaviors and show how at one level both systems are deterministic and identical, but as we move up in abstractions that consistency declines. Finally, we use packet captures to compare network behaviors and performance. We reconstruct flows and compare per-flow and per-experiment statistics.

From these comparisons, we find that the behavior of the workload in the testbeds is similar but that the underlying processes to support it do vary. The low-level network behavior can vary quite widely in packetization depending on

the virtual network driver. While these differences can be important, and knowing about them will help experiment designers, the core application and OS behaviors still represent similar processes.

Crussell, Jonathan, Thomas M. Kroeger, David Kavaler, Aaron Brown, and Cynthia Phillips. “Lessons learned from 10k experiments to compare virtual and physical testbeds.” In 12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 2019) [6].

Abstract: Virtual testbeds are a core component of cyber experimentation as they allow for fast and relatively inexpensive modeling of computer systems. Unlike simulations, virtual testbeds run real software on virtual hardware which allows them to capture unknown or complex behaviors. However, virtualization is known to increase latency and decrease throughput. Could these and other artifacts from virtualization undermine the experiments that we wish to run?

For the past three years, we have attempted to quantify where and how virtual testbeds differ from their physical counterparts to address this concern. While performance differences have been widely studied, we aim to uncover behavioral differences. We have run over 10,000 experiments and processed over half a petabyte of data. Complete details of our methodology and our experimental results from applying that methodology are published in previous work. In this paper, we describe our lessons learned in the process of constructing and instrumenting both physical and virtual testbeds and analyzing the results from each.

The following paper was supported by the SECURE Grand Challenge [12] but leveraged the simple workload and experiment scripts that were created with the support of this LDRD. For this reason, we include that paper here for completeness.

Geraci, Gianluca, Laura P. Swiler, Jonathan Crussell, and Bert J. Debusschere. “Exploration of Multifidelity Approaches for Uncertainty Quantification in Network Applications.” In Proc. of the 3rd International Conference on Uncertainty Quantification in Computational Sciences and Engineering – UNCECOMP 2019 [8].

Abstract: Communication networks have evolved to a level of sophistication that requires computer models and numerical simulations to understand and predict their behavior. A network simulator is a software that enables the network designer to model several components of a computer network such as nodes, routers, switches and links and events such as data transmissions and packet errors in order to obtain device and network level metrics. Network simulations, as many other numerical approximations that model complex systems, are subject to the specification of parameters and operative conditions of the system. Very often the full characterization of the system and their input is not possible, therefore Uncertainty Quantification (UQ) strategies need to be deployed to evaluate the statistics of its response and behavior. UQ techniques, despite the advancements in the last two decades, still suffer in the presence of a large number of uncertain variables and when the regularity of the systems response cannot be guaranteed. In this context, multifidelity approaches have gained popularity in the UQ community recently due to their flexibility and robustness with respect to these challenges. The main idea behind these techniques is to extract information from a limited number of high-fidelity model realizations and complement them with a much larger number of a set of lower fidelity evaluations. The final result is an estimator with a much lower variance, i.e. a more accurate and reliable estimator can be obtained. In this contribution we investigate the possibility to deploy multifidelity UQ strategies to computer network analysis. Two numerical configurations are studied based on a simplified network with one client and one server. Preliminary results for these tests suggest that multifidelity sampling techniques might be used as effective tools for UQ tools in network applications.

Chapter 3

Experiments

In this section, we document the unpublished work from this LDRD. This section is more technical and builds upon our published works. We have made an effort to ensure that the reader can follow the basic ideas even without understanding every detail.

Our work on oversubscription in virtual testbeds and the differences that it introduces is currently under review. We have a separate and more comprehensive write up on our calibration work that we plan to submit to a conference or archive in another SAND report.

3.1 Oversubscription in Virtual Testbeds

We have performed numerous experiments to understand the effects of oversubscription, the practice of running many virtual machines on fewer machines, on virtual testbeds. We were particularly interested in how the oversubscription changes the comparisons between virtual and physical testbeds. We conducted experiments where instead of running a single HTTP client and server on their own dedicated physical machines, we measure effects when we run an increasing number of isolated pairs, up to 80 pairs, on the same physical machines. Preliminary results show that oversubscription, and the contention that it creates, degrades the workload performance sooner than we expect. However, we have seen a performance anomaly with two or four pairs where each HTTP client instance has better throughput than a single pair, as shown in Figure 3.1 (right). We discuss these results below.

3.1.1 Exploratory Analysis

When we first designed these experiments, we expected several inflection points in the request-per-second (RPS) plots. We expected the first when the number of pairs first exceeds the number of physical cores in the system. We expected the second when the number of pairs first exceeds the number of virtual cores (with Hyperthreading enabled). We expected the third when the sum of the bandwidth of the pairs exceeded the underlying physical network speed. As we can see from Figure 3.1 (left), the virtual testbed performance degrades significantly sooner than these expected inflection points for this hardware, which has 32 physical cores, 64 virtual cores, and 100Gbps physical bandwidth (each pair is limited to

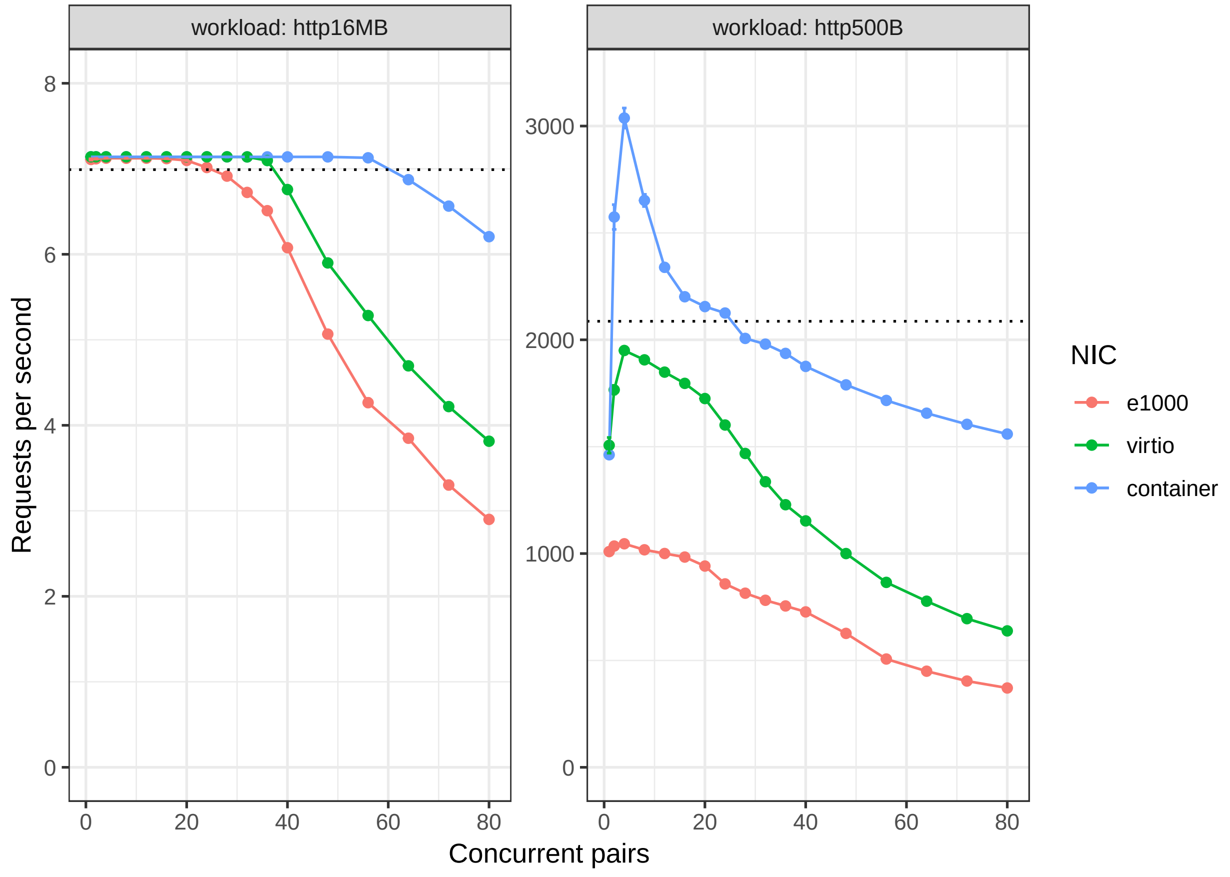

Figure 3.1. Requests per second with increasing number of concurrent pairs, split by NIC (Network Interface Card) and workload. The dotted horizontal line indicates empirical physical performance at one concurrent pair.

Table 3.1. Environment-based parameters varied for performance anomaly exploration. A ✓ indicates that we observed the anomaly while a ✗ indicates that we did not. NT represents an untested combination. See Table 3.3 for an explanation of the various parameters.

		environment setting					
	colocated / noncolocated	*hyperthreading*		*offloading*		*rate limiting*	
		on	off	on	off	on	off
network driver	*e1000*	✗/✓	✗/✓	✗/✓	NT/NT	✗/✓	NT/NT
	virtio	✗/✓	✗/✓	✗/✓	NT/✓	✗/✓	NT/✓
	container	✓/✓	✓/✓	✓/✓	NT/NT	✓/✓	NT/NT

1Gbps). We performed similar experiments on two different compute clusters with different underlying hardware and network speeds and found similar results. In future work, experiments like these could aid in the construction of virtual testbeds by helping to understand the trade-offs between purchasing more nodes, nodes with more cores, and nodes with faster networks.

The performance anomaly shows that running more than a single pair results in higher per-pair performance than when running a single pair. When designing the experiments, we expected that the RPS would be monotonically decreasing; as we added pairs to the system the per-pair performance would remain the same or decrease. As we can see in Figure 3.1 (right), running several pairs results in a higher per-pair performance. We spent a significant amount of time investigating this performance anomaly, running many experiments to see if it was dependent on certain environmental settings (such as Hyperthreading and rate limiting) and experimental settings (such as the virtual network interface). During these experiments, we tried many different forms of instrumentation in order to better understand the cause of the performance anomaly. Specifically, we looked at fine-grain timing in *ApacheBench*, interrupt behavior, KVM exit events, and network events in the Linux kernel. We were unable to pinpoint the underlying cause of the anomaly, however, we did find a correlation with the occurrences of Linux kernel network events in the host kernel. A summary of experiments performed and metrics collected can be found in Tables 3.1 and 3.2, respectively.

We omit the full details of these experiments as these are contained in a paper that is under review.

3.1.2 Regression Analysis

As we have many testbed parameters that vary across increasing levels of contention, an extensive independent analysis for each set of parameters would be tedious and not allow us to analyze how parameters interact. Thus, we require an analysis method that can help us

Table 3.2. Instrumentation-based metrics added for performance anomaly exploration.

Instrumentation Class	Metric Description	Evaluated Metrics
System Interrupts	Difference in counts from */proc/interrupts* before and after experiment run.	153
Tracing	KVM event time tracing: *mmio*, *vmexit*, and *ioport*; three fields (*min*, *max*, *average*, and *percent time*) each.	12
ApacheBench connection times	Time spent in each connection phase: *connect*, *waiting*, *processing*, and *total*.	4
TCPTrace	Connection throughput across all flows in a given experiment.	1
Network Performance	Commands sent over the network: interfaces *eth2*, *mega_bridge*, and *mega_tapX*; 6 command types.	18
	Total:	188

understand the effects of these parameters on our metrics of interest. In analyzing requests per second, we use ordinary least squares (OLS) linear regression. This allows us to inspect the relationship between our response (*dependent variable*) and our explanatory variables of interest (*predictors* or *covariates*, *e.g.*, colocation enabled). All predictors in this model are testbed parameters that can be varied. A description of each variable can be found in Table 3.3. The hardware used in each compute cluster are as follows:

Cluster 1:

- Dual Intel®Xeon®E5-2630L 0 @ 2.00GHz
- 24 Cores (6/socket, HT enabled), 125GB Memory
- Interfaces: 1Gbps (igb), 10Gbps (ixgbe)

Cluster 2:

- Dual Intel®Xeon®E5-2630 v3 @ 2.40GHz
- 32 Cores (8/core, HT enabled), 377GB Memory
- Interfaces: 1Gbps (igb), 40Gbps (mlx4)

Cluster 3:

Table 3.3. Testbed parameters varied for regression models.

Testbed Parameter	Description
Concurrent Pairs	Number of concurrent workers used in a given experiment
Compute Cluster	Physical compute cluster used (C1, C2, or C3)
NIC	Network interface used (e1000, virtio, container)
Instrumentation	Whether or not kernel instrumentation is enabled (e.g., to track system calls precisely), introduces overhead
GRE	Whether GRE tunnels are used to trunk packets between physical machines or VLAN tagging is used instead
Colocation	Whether client and server pairs were on the same physical machine or not
Hyperthreading	Whether or not Hyperthreading is enabled across CPUs in the testbed

- Dual Intel®Xeon®E5-2683 v4 @ 2.10GHz
- 64 Cores (16/socket, HT enabled), 503GB Memory
- Interfaces: 10Gbps (ixgbe), 100Gbps (mlx5)

When performing regression modeling, one is interested not only in standard "goodness-of-fit" measures (*e.g.*, R^2), but also the results of model diagnostics to avoid, *e.g.*, heteroscedasticity and multicollinearity [2], which can severely alter model fit and thus threaten statistical robustness. We take great care to make sure that our models pass these diagnostics as much as possible given available data, and thus can be inferred from. To check for heteroscedasticity, we inspect residual and normal qq-plots; for multicollinearity, we calculate the variance inflation factor (VIF) [4] *prior* to introducing higher-order (interaction) terms, as calculating VIF with higher-order terms does not achieve its intended purpose (lower-order terms are intrinsically correlated with their higher-order counterparts). We report that all pre-higher-order VIFs lie below standard guidelines (5). In addition, we *log*-transform the requests per second to stabilize variance and improve model fit [4], as suggested by the Box-Cox procedure [3].

We chose to model only the http16MB workload, as we were unable to find the root cause of the performance anomaly described above. We could have chosen to account for this anomaly by, *e.g.*, adding a dummy variable to represent the curve, but this puts the interpretation of the other coefficients in jeopardy; we may overfit to the observed curve, and thus influence the values of coefficients based on real values, altering their interpretation. We leave this fitting to future work, once the cause of the anomaly is identified.

Table 3.4. Requests per second OLS regression (http16MB workload).

	Dependent variable:
	Log Requests Per Second
Concurrent Pairs^2	-0.0002^{***} (0.00001)
Concurrent Pairs	-0.034^{***} (0.0005)
Compute Cluster C2	-0.066^{***} (0.011)
Compute Cluster C3	-0.036^{***} (0.008)
NIC virtio	-0.070^{***} (0.005)
NIC container	-0.176^{***} (0.006)
Instrumentation enabled	0.090^{***} (0.005)
GRE enabled	-0.076^{***} (0.010)
Colocation enabled	-0.052^{***} (0.005)
Hyperthreading enabled	-0.017^{***} (0.005)
Concurrent Pairs * Cluster C2	0.007^{***} (0.0004)
Concurrent Pairs * Cluster C3	0.023^{***} (0.0003)
Concurrent Pairs * NIC virtio	0.017^{***} (0.0002)
Concurrent Pairs * NIC container	0.024^{***} (0.0002)
Concurrent Pairs * Instrumentation enabled	-0.015^{***} (0.0002)
Concurrent Pairs * GRE enabled	0.003^{***} (0.0004)
Concurrent Pairs * Colocation enabled	-0.0004^{**} (0.0002)
Concurrent Pairs * Hyperthreading enabled	0.005^{***} (0.0002)
Constant	2.093^{***} (0.009)
Observations	80,286
R^2	0.733
Note:	*p<0.1; **p<0.05; ***p<0.01

Table 3.4 shows the results of our regression. Each parameter is interacted with concurrent pairs to examine the change in effect of, *e.g.*, using containers, as contention increases. Categorical variables are interpreted with respect to their base levels: C1 for compute cluster, e1000 for NIC, instrumentation disabled, GRE disabled, colocation disabled, and hyper-threading disabled. We center the concurrent pairs variable at 1 in order to have interpretable lower-order terms; without centering at 1, lower-order terms would have to be interpreted as estimated with 0 concurrent pairs, which is impossible.

To interpret these results, consider the case of comparing requests per second for containers compared to e1000 at 40 concurrent pairs. The general form of the regression equation (using expected values) with an interaction between X_i and X_j for some $i < k, j < k, i \neq j$ and a logged dependent variable (Y) is:

$$log(Y) = \beta_0 + \beta_1 X_1 + \dots + \beta_i X_i X_j + \dots \beta_k X_k$$

With all other variables held constant, the difference in log requests per second for containers *vs.* e1000 is:

$$log(Y_{containers}) - log(Y_{e1000}) = -0.176 + (40 - 1)(0.024) = 0.776$$

We can interpret this as a percent change by realizing:

$$\begin{aligned} log(Y_{containers}) - log(Y_{e1000}) &= log(\frac{Y_{containers}}{Y_{e1000}}) \\ &= log(1 + \frac{Y_{containers} - Y_{e1000}}{Y_{e1000}}) \end{aligned}$$

Thus, the percent change in requests per second due to using containers with 40 concurrent pairs, *with all other variables held constant and controlled*, is $e^{0.776} - 1$, or 117.314%. A similar calculation can be performed to interpret all other coefficients in the model

3.1.3 Detecting Contention

Oversubscription can cause contention but oversubscription is required to emulate large system. We investigated whether we could instead allow oversubscription but detect contention so that we can invalidate the result of the experiment. Figure 3.2 depicts the relationship between our host utilization metrics and requests per second. The benefit of the host metrics are that they are low cost and easy to collect without directly instrumenting the VM. The figure shows the empirical second derivative - a representation of concavity. Thus, each depicted inflection point represents a change from decreasing to increasing slope over increasing pairs.

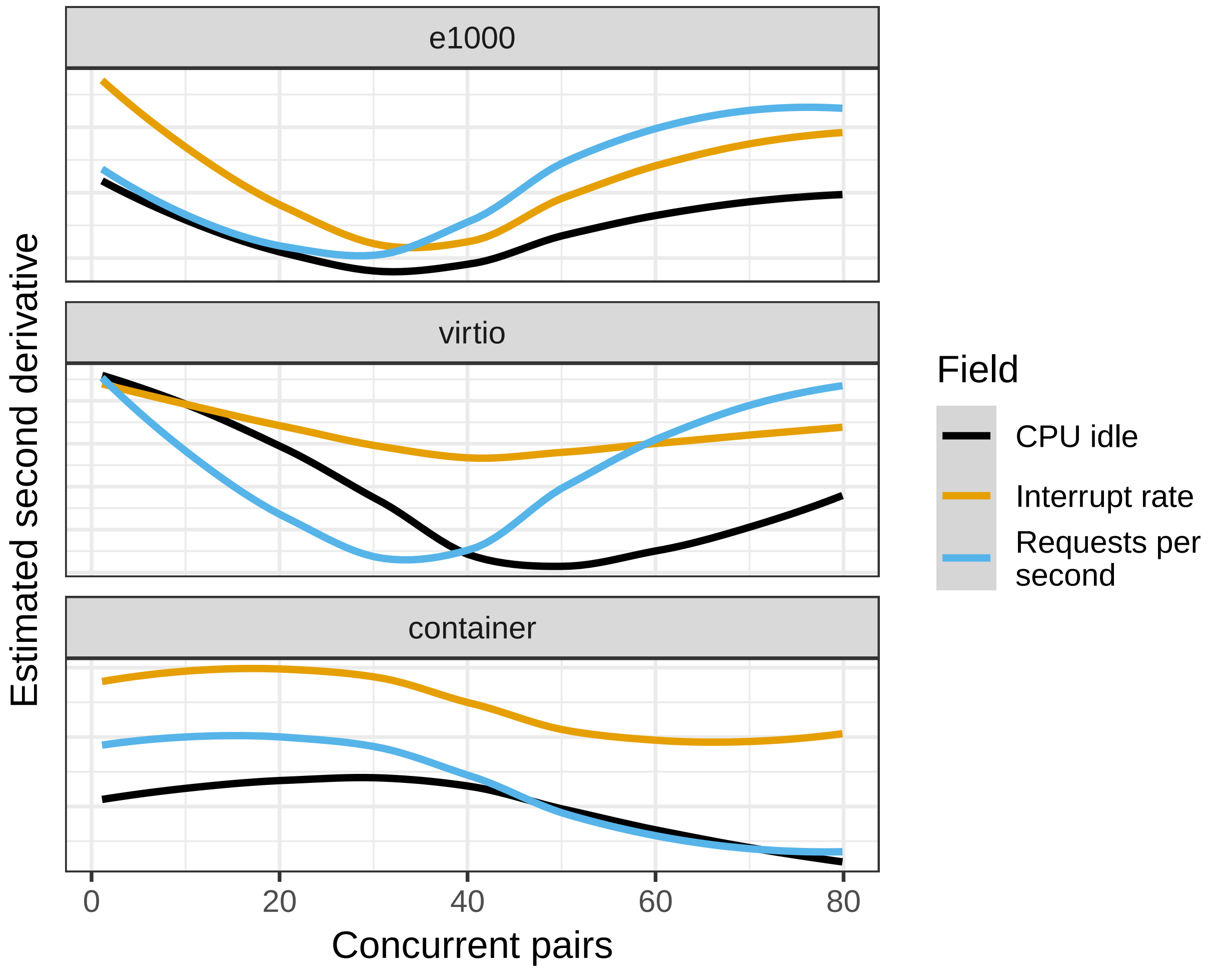

Figure 3.2. Comparison between host utilization metrics and requests per second for the http16MB workload, split by network interface for the follow testbed parameters: no colocation, offloading enabled, and instrumentation disabled (see Table 3.3). CPU idle y-values multiplied by -1 to facilitate visual comparison.

We calculate these values by LOESS smoothing [7] the original value (*e.g.*, requests per second), then approximating the second derivative by calculating $\frac{\Delta^2 y}{\Delta x^2}$ between successive concurrent pairs. As we are interested in comparing relative values rather than absolute, the calculated second derivatives are scaled to be within the same range (hence the lack of a y-axis unit). The y-value for CPU idle is multiplied by -1 (*i.e.*, reflected about the horizontal axis) to facilitate visual comparison.

We see that the inflection points for requests per second consistently precede that of CPU idle time and interrupt rate. In addition, we see a similar general shape (with somewhat differing magnitude) for each field. Further research will be required to determine if this technique can be used to detect contention in virtual testbeds.

3.2 Calibrating Results from Virtual Testbeds

To begin to understand how to calibrate results from virtual testbeds using results from physical testbeds, we ran a set of experiments on each and used various regression models to fit the output from virtual testbeds to a predicted physical result. We reuse our simple experiment scenario: an HTTP client loading fixed-sized pages from an HTTP server. In this scenario, we parameterize the number of threads that the client uses to make requests and the size of the response from the server (the payload size). Our calibration task is as follows: can we predict the requests per second (RPS) for a physical testbed using the RPS measured in a virtual testbed. We utilize 4,000 results from virtual testbeds and 400 results from physical testbeds, described below.

We conducted a total of 4,000 experiments on the virtual testbed, 2,000 each for two virtual network types: e1000 and virtio. We sampled 50 payload sizes between *500B* and *16MB* and ran across four client thread counts: 1, 2, 3, 4. To ensure that we had more weighting towards smaller flows which are more prevalent in the real world, we uniformly sampled the *log* of the payload sizes. For each sample, we ran a total of ten tests to measure variability.

We conducted a total of 400 experiments on the physical testbed. From the 50 payloads described above, we randomly selected 20 payloads to test on the physical testbed. We ran these across the same four client thread counts. Since physical testbed experiments are more expensive, we only ran five iterations per sample.

The raw RPS values across the three environments (physical, e1000, and virtio) are depicted in Figure 3.3. The number of client threads results in some stratification across the lower payload sizes but all environments and numbers of client threads converge towards the same limit. When we split the data by client thread count, we find that the data appears to follow a logistic curve so we can apply non-linear regression methods to model a prediction interval for the RPS for each thread count in each environment, as shown in Figure 3.4 for one client thread. We were unable to derive models that used both the number of client threads and the payload size to predict the RPS.

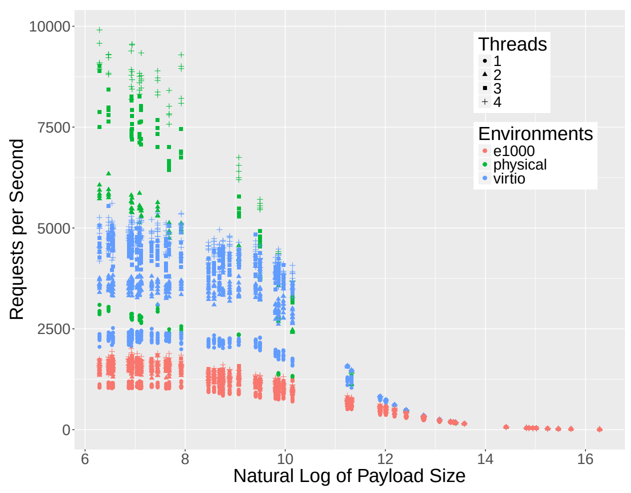

Figure 3.3. Requests per second across payload sizes and thread counts for all three tested environments.

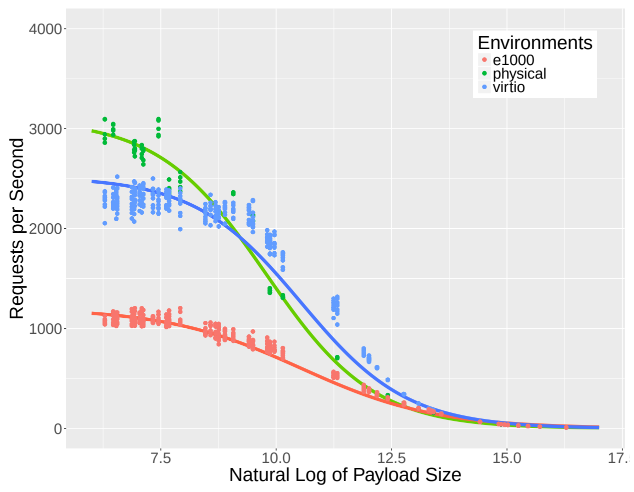

Figure 3.4. Requests per second non-linear regression models for a single client thread across the three tested environments.

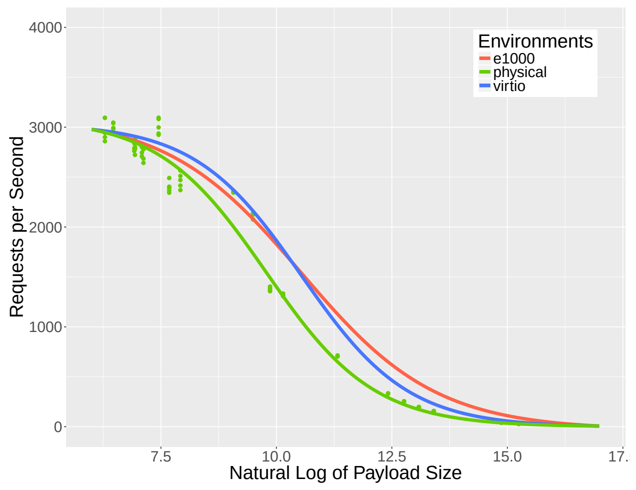

Figure 3.5. Requests per second equated models for e1000 and virtio as compared to non-linear regression for physical testbed for a single client thread.

From these initial findings, we have developed models to equate the per-thread count models between environments as shown in Figure 3.5. From this model, we can predict the RPS values for the physical testbed using the measured e1000 or virtio RPS. We can also use our previous models for the RPS for e1000 or virtio to predict the RPS for the virtual testbeds for unknown payload sizes and then use the equating model to predict the RPS for the physical testbed.

The equated models allow us to measure the difference between expected RPS and observed RPS for the physical testbed to determine the prediction error for the different virtual testbeds for individual payload sizes. Additionally, we can measure the area between the curves for the predicted RPS for the equated e1000 and virtio models and the regression model for the physical testbed. The area between the curves could be a useful measure of the relative error between the e1000 and virtio models, suggesting which should be preferred for estimating the RPS.

We omit the full details of these experiments and analysis as we plan to prepare a paper on these results.

3.3 Generalizing Background Workload

There are many different workload sizes and types that we can test beyond http50B and http16MB. In lieu of exhaustively testing a multitude of workload sizes and types, we attempted to represent these potential workload differences by using the Linux *stress* tool. By utilizing *stress*, we can simulate arbitrary artificial workloads by putting stress on various

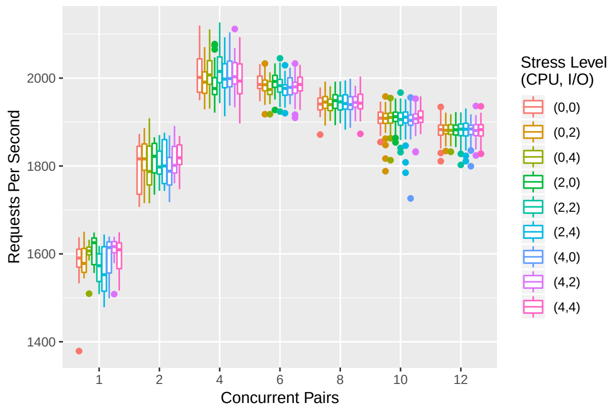

Figure 3.6. Stress level comparison for increasing contention, http50B workload, *virtio* NIC.

parts of the system, which is hopefully representative of real (differing) workloads.

The *stress* tool enables experimenters to impose simulated stress based on CPU, memory, and I/O workloads. We tested the effects of stress levels $0, 2$, and 4, corresponding to the number of workers which are used to simulate aforementioned stress. Figure 3.6 illustrates the effect of CPU and I/O stress for increasing contention (concurrent pairs); we omit memory stress and the http16MB workload for brevity (results were analogous).

As shown, we see no general effect of stress on requests per second. To verify this across all experimental parameters (*i.e.*, offloading, various NICs, Hyperthreading, *etc.*), we additionally fit a regression in order to determine whether or not stress has a significant effect on requests per second while controlling for all other measured parameters; we saw no such significance (regression coefficients omitted for brevity).

The reason behind the lack of a detected significance may be due to the *stress* utility not being an accurate representation of real-world stress. Or, the system could be robust to the types of stress that the utility can simulate. In the future, one can try many different artificial workloads - beyond http16MB and http50B - instead of attempting to simulate these workloads; this would help determine whether or not *stress* is an accurate representation of real life scenarios, and also provide additional information as to where virtual and physical systems differ.

3.4 Security Monitoring

As a part of our efforts to determine the differences between virtual and physical testbeds, we ran Zeek [11], a security monitoring tool, across the network traffic generated by our physical, e1000, virtio, and container experiments. We did this for both the http500B and

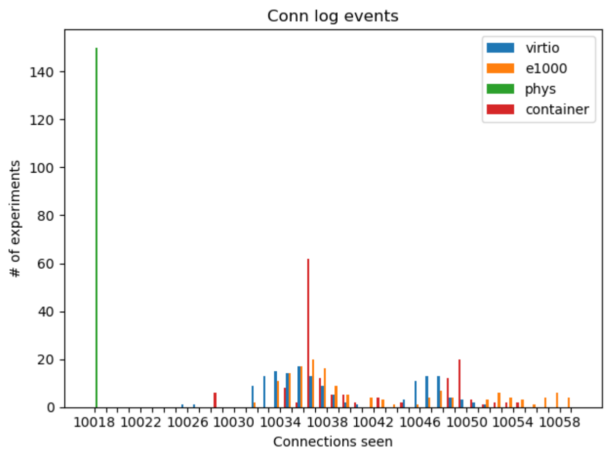

Figure 3.7. Distribution for the number of connections seen per experimental run for the 500 byte workload.

http16MB workloads.

Initially, Zeek found many checksum errors when trying to process the packet captures from our virtual testbeds tests. We hypothesize this was the result of TCP offloading aggregating multiple packets together and not recomputing the checksums. The fact that this was far more frequent in the large payload helps to support that hypothesis.

Rerunning our tests and telling Zeek to ignore invalid checksums produced more complete results but still resulted in significant differences in their behaviors across workloads and platforms (physical, e1000, virtio and container). We ran each of experiment three times in order to measure the variability of our results.

We first looked at the total number of connections seen by each experiment. Figure 3.7 and Figure 3.8 show the distribution of the number of connections seen in a given experiment, normalized over the number of times each experiment was run, for the http500B and http16MB workloads, respectively. From this figure, we can see that the physical testbed consistently reproduced the same result. The other platforms have a wide range of connection counts that is above the baseline set by the physical system. We hypothesize that these environments add several extra connections that appear as noise. It is interesting to note that the absolute number of additional connection is similar across payload sizes. Specifically, the virtual testbeds had connection counts that were 10-50 more than the physical testbed, independent of the workload.

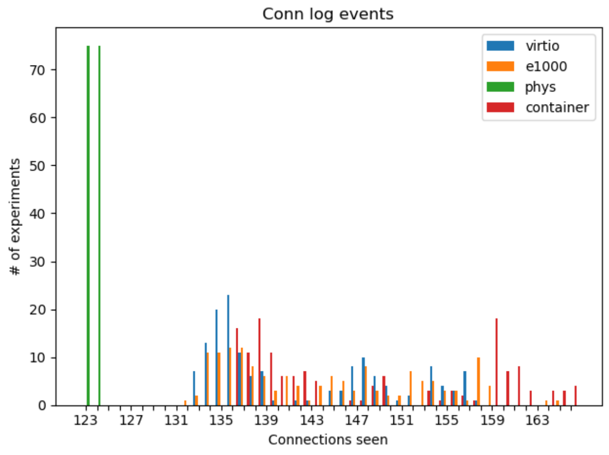

Figure 3.8. Distribution for the number of connections seen per experimental run for the 16 MB workload.

3.5 Network Fingerprinting

As a part of our efforts to determine the differences between virtual and physical testbeds, we ran traffic fingerprinting tools *nmap*, *p0f*, and *HTTPRecon* to assess whether the different environments produced different fingerprints. Specifically, the goal was to see if virtualization has any effect on the results returned by fingerprinting tools that try to identify the remote system. We find that there is no apparent difference in the output of fingerprinting tools when used on virtual versus physical testbeds.

We replicate our simple test environment, an HTTP client and server that are connected on the same local network, with a few small variations for these tests. Instead of running a regular HTTP client, the client used *nmap* to try to fingerprint the server's OS and HTTPRecon to fingerprint the webserver running on the HTTP server. Instead of using *protonuke*, the HTTP server runs Apache 2.2.6 webserver. The reason to use such an old version of Apache is because it is the latest version fully supported by HTTPRecon. The server also uses the passive fingerprinting tool *p0f* to try to fingerprint the client machine.

We found that the tool output from both the virtual and physical testbeds was identical. The only differences of note were the timestamps, MAC addresses, and latency which would differ on other physical or virtual testbeds as well.

Chapter 4

Future Experiments

We conducted and analyzed many experiments during the LDRD. In this section, we detail the various experiments that we would have run, if we had the bandwidth to do so. These experiments would provide researchers with a more comprehensive understanding of the implications of various choices on virtual testbeds.

Virtual NICs We primarily conducted experiments with e1000 and virtio-net-pci virtual network interfaces. There are many other virtual network interface types available in QEMU including e1000e and vmxnet3 which would be interesting to study in future work.

Additionally, network interface passthrough (such as *sr-iov*) could allow the same network drivers used by the physical nodes (typically *igb* or *ixgbe*) to be used within VMs. This could allow future work to more directly compare behaviors. Understanding how this passthrough would perform on nodes with many virtual machines would also be useful to do in future work.

Hardware settings We conducted experiments with Hyperthreading enabled and disabled which required modifying the system BIOS. There are other hardware settings such as NUMA and CPU power and performance policies to explore.

Heterogeneous environments Our experiments used the same virtual network interface on both VMs. Future work could compare the behaviors of VMs with differing network interfaces. Furthermore, future work could study the mixing of VMs and containers.

Isolated Contention In our contention experiments, we used equal numbers of servers and clients where VMs (or containers) had equal contention. Future work could explore varying the contention for different VMs in the workload to better approximate realistic testbed experiments. For example, experimenters could have all servers running on a single node and vary the number of physical nodes that the clients are split among to both extremes: all on one node or all on separate nodes.

Partial Instrumentation In our experiment harness, instrumentation is all-or-nothing. Given the high overhead of instrumentation, future work could explore whether the same information can be gleaned from instrumenting fewer than all of the VMs (or containers).

Varying Topologies To start simple, our experiments included a single client and server on the same local network. Future work could study a more complicated topology where the client and server are separated by a network of routers with vary topology. Note that this could potentially involve re-cabling a physical testbed in order to vary the physical topology.

Hybrid testbeds Future work could study how testbeds with some physical and some virtual components compare to their purely virtual or purely physical counterparts. There could be many permutations of such a hybrid testbed with trade offs between which components are physical and which are virtual.

Other workloads We conducted experiments primarily with HTTP as our representative workload with some limited tests with HTTPS. There are many other client-server protocols that may have differing characteristics between virtual and physical testbeds. Furthermore, there are many distributed protocols that could be interesting to study in future work.

Virtual Switch We used Open vSwitch as the virtual switch in our virtual testbeds. There is a DPDK-enabled version of Open vSwitch which offers better performance but potentially introduces different artifacts. Furthermore, there are alternative virtual switches which could be used instead of Open vSwitch.

In the clusters we tested on, we VLAN-tag packets to pass along to the physical switch. In some clusters, VLAN switching is not available so instead experimenters rely on layer-3 tunnels such as *GRE* between hosts. In other clusters, the switches support *VXLAN*. Understanding how these different methods to connect VM networks behave is an important area for future work.

Hypervisor We used KVM as the hypervisor in our virtual testbeds. There are many other hypervisors such as Xen and VMWare that may introduce different artifacts that could be tested in future work. Furthermore, there are various parameters that can be tweaked within the hypervisor such as the scheduling algorithm that could be explored in future work.

Software versions We conducted experiments using a fixed set of software versions to increase reproducibility and limit the chances of artifacts being introduced by differing software versions. Future work could explore how results change over software versions or with mismatched software versions.

References

[1] Emulytics: Emulation, modeling, and analysis. `https://www.sandia.gov/emulytics/`, 2019.

[2] Galen Bollinger. Book review: Regression diagnostics: Identifying influential data and sources of collinearity, 1981.

[3] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[4] Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press, 2014.

[5] Jonathan Crussell, Thomas M Kroeger, Aaron Brown, and Cynthia Phillips. Virtually the same: Comparing physical and virtual testbeds. In *2019 International Conference on Computing, Networking and Communications (ICNC)*, pages 847–853. IEEE, 2019.

[6] Jonathan Crussell, Thomas M Kroeger, David Kavaler, Aaron Brown, and Cynthia Phillips. Lessons learned from 10k experiments to compare virtual and physical testbeds. In *12th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 19)*, 2019.

[7] John Fox. Nonparametric regression. *Appendix to: An R and S-PLUS Companion to Applied Regression*, 2002.

[8] Gianluca Geraci, Laura P. Swiler, Jonathan Crussell, and Bert J. Debusschere. Exploration of multifidelity approaches for uncertainty quantification in network applications. In *Proc. of the 3rd International Conference on Uncertainty Quantification in Computational Sciences and Engineering–{UNCECOMP}*, 2019.

[9] minimega developers. minimega: a distributed vm management tool. `http://minimega.org/`, 2019.

[10] Ronald G Minnich and Don W Rudish. Ten million and one penguins, or, lessons learned from booting millions of virtual machines on hpc systems. In *Workshop on System-level Virtualization for High Performance Computing in conjunction with EuroSys*, volume 10, 2009.

[11] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[12] Ali Pinar and Zach Benz. Science & engineering of cyber security by uncertainty quantification and rigorous experimentation. `https://securegc.sandia.gov/`, 2019.

[13] S. S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946.

DISTRIBUTION:

1	MS 0899	Technical Library, 9536 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office, 1911