USC Viterbi
School of Engineering
Information Sciences Institute

DETER Technical Report
ISI-TR-683
2011

Building Apparatus for Multi-resolution Networking Experiments Using Containers

Authors: DETER Team

This material is based upon work supported by the Department of Homeland Security, and Space and Naval Warfare Systems Center, San Diego, under Contract No. N66001-10-C-2018.

Any opinions, findings, and conclusions or reccommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Deparment of Homeland Security for the Space and Naval Warfare Systems Center, San Diego.

USC Information Sciences Institute • 4676 Admiralty Way • Marina del Rey, CA 90202

Building Apparatus for Multi-resolution Networking Experiments Using Containers

Abstract

We describe a system approach and key abstraction intended to facilitate the modeling and emulation of highly scaled, complex, *multi-resolution* networking problems within the DETER testbed. Multi-resolution problems are those that are best modeled within a testbed by representing different aspects of the overall scenario at different scale and fidelity. The present work facilitates structured and efficient support of such problems within our testbed, allowing each individual component of a scenario to be modeled at precisely the required fidelity while the whole scales efficiently to very large size. Our key abstraction is the *container*. Different types of containers each contain experimental elements embedded at different levels of abstraction, while providing a uniform interface and self-description of characteristics. Containers can hold elements or other containers, and define the communications scope of their contents. We describe the concept, design and a prototype implementation, including a demonstration that instantiates a scenario of more than 650 elements on 8 computers.

1 Introduction

We propose a system to enable experimentation on large, multi-resolution, networking problems. A multi-resolution problem is best understood by representing its different aspects at different fidelity. As the Internet gets larger and more complex, network researchers have begun to investigate multi-resolution phenomena. For example, the spread pattern of a zero-day worm across the whole Internet can affect the properties of a distributed denial of service attack that the worm's payload delivers. The worm propagation is best understood using contagion models that represent large parts of the network abstractly. Evaluating the effect of the attack depends on modeling a few interconnected systems in great detail.

Wide-area DNS cache poisoning is a another multi-resolution problem. Cache poisoning opportunities arise from user behavior, modeled coarsely, and success depends on cache race conditions, modeled finely. Our system encourages experimentation on such phenomena by representing its components at appropriate levels of abstraction in an experimental *apparatus*. An apparatus is a controlled setting containing the experimental tools and monitoring systems necessary for carrying out repeatable experiments.

Current network testbeds cannot directly represent the multi-resolution phenomena that interest us, though they are natural platforms in which to build an apparatus. Testbeds [2, 14, 21] have made direct experimentation on the order of a few hundred computers more accessible and useful to network researchers. Some systems provide tools for virtualization[12, 13] or integrated simulation[11] to scale experiments up or to import abstractions, but they supply the researcher little guidance in which to choose or how to apply them. A researcher who wants to use those tools must largely depend on their own knowledge and experience.

Our insight is to use the abstractions that arise naturally from modeling and thinking about a multi-resolution problem to guide the construction of an apparatus where specific components maintain the fidelity the experimenter requires and the whole scales efficiently. If an experimenter needs a network that reports the progress of a worm propagating based on a contagion model at the granularity of network location, details of how the worm infects each host can be omitted and a larger network modeled.

This paper addresses the problem of converting the researcher's conceptual apparatus into a real apparatus that is scaled and abstracted appropriately to the researcher's goals. The conceptual apparatus includes the general topology annotated with fidelity requirements. From that our system generates a real apparatus meeting those requirements.

The fundamental building block of our system is the *container*, which represents experimental elements at the same level of abstraction; we build each apparatus from containers. Each container collects similar resources and can make those resources available as either concrete elements or as part of nested containers that hold simpler elements.

A container also provides a communications scope for the elements inside it, allowing elements to communicate across abstraction levels. Elements in the same container can talk directly, and elements talking between containers have their communications translated at the well-defined container boundaries.

Containers characterize their abstraction level in a way meaningful to the researcher, implement a useful range of implementation models/abstraction levels, and present a standard interface for combining elements across levels of abstraction.

Because using containers highlights concrete scaling/abstraction trade-offs, researchers are guided to construct a useful, realizable apparatus. When using containers to realize an experiment the scaling properties and effects on abstraction are explicit, meaning that researchers need not be experts in scalable implementation strategies to produce meaningful multi-resolution experiments. As we understand the problem more, the tools will do more of this analysis for the researcher.

The standard interfaces of containers accelerate integration of new scaling technologies. New techniques for virtualization, simulation, and other scaling approaches appear frequently; once containerized they express their tradeoffs explicitly and can be configured directly.

The system combines containers into a practical environment using standard interfaces. Once a conceptual apparatus is mapped into containers appropriate for the experiment being conducted the apparatus must be realized. That is, the containers need to be created, concrete elements they contain need to be loaded with software, and the elements must be interconnected. Providing a minimal interface to containers that spans multiple abstraction levels is a key challenge.

We have designed the container interfaces and prototyped the tools needed to create experiments from them in the DETER testbed. DETER is both an open Emulab-based testbed and a research project investigating principled methods of network experimentation[14].

We tested our prototype by constructing an instantiation of the worm/DDoS experiment outlined above that consisted of more than 650 concrete elements instantiated on 8 physical machines communicating between three abstraction technologies.

The remainder of this paper describes our container design and its relationship to other work, and then describes our prototype implementation.

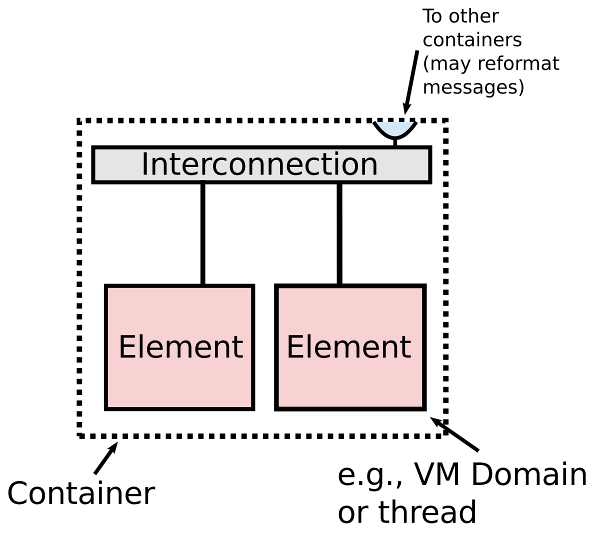
To other containers (may reformat messages)
Interconnection
Element
Element
Container
e.g., VM Domain or thread

Figure 1: A container holding two concrete elements

2 Containers

Containers form the backbone of our apparatus building system. Each container represents a collection of similar resources that can be used to construct the apparatus. A container is conceptually full of computers, processes or some other implementation platform. We precede the word “container” with the resource type, so we refer to “computer containers” or “process containers.” The system builds apparatuses from those resources by allocating and connecting them hierarchically through standard container interfaces. Domain-specific tools assist can in this process.

A researcher can dedicate a container’s resources directly to the apparatus or to supporting a nested container. A computer container has interfaces to allocate computers directly, or to use computers as virtual machine containers for lower fidelity types of elements, such as virtual machines or processes. Similarly, a builder could ask the same computer container to create a container of Hadoop tasks out of multiple computers.

When resources are allocated to an apparatus directly we call them concrete elements. Concrete elements in the same container represent the same type of element. In addition to allocating these elements, the researcher specifies the interconnections between them and interconnection points that allow concrete elements to communicate outside the container. When communications leave or enter the container, they are translated to or from the container’s representations. Packet events from a simulation are translated into packets on a wire by those interconnection points. A container holding two concrete elements is depicted in Figure 1.

A container also knows what kind of containers it can hold. For one container to hold another, there are two requirements. First, the inner container must be able to use the outer container's resources. A computer is not built from virtual machines, so a computer container cannot be created inside a virtual machine container. Second, the inner container must be able to translate to and from the outer container's communication model.

Each container expresses its scalability and abstraction level. Because abstraction is multi-dimensional, the metrics we use depend somewhat on the container technology. This scaling information includes constraints. A simulation-thread container can advertise the ability to create up to 1000 threads with packet rates of (90,000 - 500 x total threads) packets/second.

For specialized containers, picking the units of a scaling expression is straightforward. A discrete event simulation can support a given event rate or a traffic generator a given packet rate. For researchers already expressing their abstractions as event or packet rates, these descriptions are meaningful; for researchers thinking in different terms these abstractions are incompatible.

More general platforms, such as physical computers or virtual machines are more difficult to characterize because their computational power or I/O bandwidth can be used in many ways. However, we believe that such adaptable concrete elements will be used primarily when higher fidelity is needed. In that case, the real-world characterization of the systems is the appropriate abstraction.

The container model captures a broad range of apparatus constructors, including Emulab-like computer-based testbeds, simulation systems, and virtual machine monitors. The nesting abilities captures the process of allocating resources of one kind to act as the substrate for another. Fixing each technology to a standard interface allows our system to express complex apparatus configurations and to add new technologies to the system as they appear.

2.1 Topology Building

Once an apparatus is described in terms of containers, the system must build it. Because the elements of the apparatus are all in containers, the researcher constructs the apparatus by asking a container to create it. In practice this request is made on a computer container, which is a testbed like DETER that understands the container interface.

We describe apparatus topologies using an extension of the topology description language (topdl)[8] used by DETER Federation[9]. Topdl is a declarative, extensible language that expresses the interconnections between network elements. We annotate network elements with

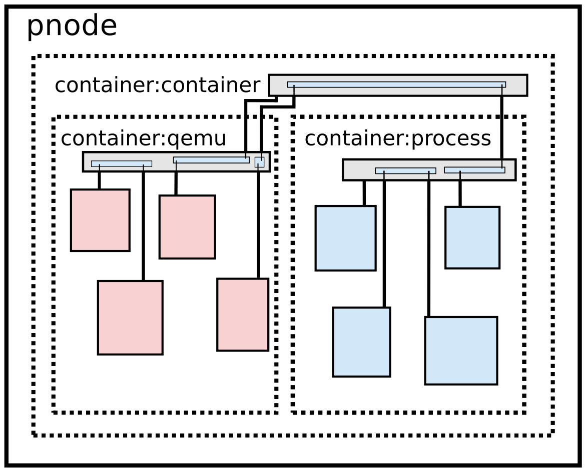

Figure 2: Multiple containers and elements demonstrating the use of a container container

the resource type that should be used to instantiate them. We discuss how such a topology can be created in Section 2.3.

When the researcher asks the outer container to create the topology, the container splits the topology into elements it can directly realize and containers it supports. It directly realizes and interconnects the concrete elements and creates containers to hold the others. The container then requests the sub-topologies (and inter-container connection points) from the inner containers. If any container fails to allocate based on its capacity or compatibility, the allocation fails.

Figure 2 depicts such a multi-container topology. The two inner containers each have realized several concrete elements. The process container holds several process level objects and the QEMU[4] container holds virtual machine instances. Communications within the containers are coordinated as in Figure 1. Communications between the containers are managed by the container container that encloses them.

A container container can only hold other containers and defines a communication scope for the inner containers. Communications within the container container have a simple format that many containers can export. When communications leave the container container it is responsible for the translation. A container container is used when several kinds of concrete elements are operating on the same physical resource.

2.2 Using the Apparatus

Once the concrete elements have been instantiated and connected, they are manipulated by standard interfaces to carry out the experiment. Although concrete elements represent a wide range of implementation possibilities the interface to manipulate them is simple, though some of the parameters depend on the elements in use.

A concrete element can be started, stopped and loaded. Loading an element puts it into its initial state, starting it begins its operation, and stopping to stops its experimental functions. Some concrete elements can be stopped and restarted, while others revert to their initial state on restart.

Loading a concrete element has different effects and takes different inputs for different elements. Loading a simulation thread can be a matter of passing it a short configuration; loading a computer can be a matter of installing an operating system and configuring an application suite.

We mitigate these differences in two ways: by specifying standard container interfaces where possible and leveraging the use of cross platform tools. Where variations on an implementation strategy are common we define or adopt unified standards. Rather than assuming that researchers will be conversant with many forms of virtual machine configurations, we adopt standards such as libvirt[5]. Similar opportunities to use standard disk image formats are available.

In addition to adopting widely applicable standards and conventions within the container interface, we also encourage the use of emerging cross-platform configuration and experiment management tools. Stork[6] can load software many instances and types of virtual machine efficiently. Plush[1] can configure and control many physical and virtual machine instances. SEER is a an experiment control platform that manages experiment configuration and operation across many platforms[18]. All these tools currently configure and control a range of virtual and physical devices, and all are designed for extension to new environments.

Making use of both the evolving scalable experimentation tools, and the careful extension of the containers interfaces, we configure and operate large scale experiments.

2.3 Embedding Topologies

As an apparatus progresses from the researcher's conception to a collection of resources on which an experiment can be carried out, there are two points where the container model influences the conceptual to concrete mapping: when conceptual resources are assigned to containers and when containers realize concrete elements. The system provides guidance at each point.

When elements are being assigned to containers, the system guides the researcher in two ways. First the abstraction advertisements restrict the choice of containers to those that match the researcher's model. Second, the scaling information guides both which container types are candidates and how many instances of an element to put in a container due to the scaling constraints. While this constraint matching problem can be complex, we expect the fact that the largest number of elements will usually be at a coarse degree of abstraction, and therefore loosely constrained, will keep it manageable.

The second problem is assigning the concrete elements to physical realizations. At some point the virtual machines need to be instantiated, the simulators loaded, and the computers allocated. This can involve the placement of a large number of elements. We take advantage of the hierarchical structure of the container layout to guide an recursive embedding process.

As in the allocation of federated sub-experiments in the DETER testbed[9], the container system breaks the allocation problem into sub-problems, and passes those sub-problems to domain-specific embedders. Unlike the general problem of breaking a network topology into components that the DETER federator faces, the container layout problem is organized by the researcher's abstraction and the hierarchical organization of components. This breaks the problem into largely independent sub-problems amenable to individual embedders. When multiple VMs are laid out on multiple computers, the scalability constraints have made each VM layout tractable as well as limiting the inter-computer bandwidth.

3 Related Work

We discuss three areas of work related to our system: other experiment apparatus creation systems, systems that offer abstractions of network elements that may be the basis for future containers, and systems related to the loading and management of concrete elements.

The most successful network experimentation platforms are Emulab and PlanetLab[2, 21]. Both have years of operational experience offering automatic creation of experimental apparatus. The two facilities have different focuses; Emulab provides a set of tightly controlled computers and configurable networks that can be closely monitored and PlanetLab provides access to services on virtualized computers throughout the Internet. These lines blur in that PlanetLab can support dedicated connections using the VINI system for allocating network bandwidth[3] and Emulab can modify its network emulation based on PlanetLab measurements using the FlexLab system[16].

Of the two systems Emulab is more compatible with our model of creating experimental apparatus for controlled experimentation. Emulab provides both virtualization[12] and integrated simulation[11] services in an attempt to construct larger apparatuses. Unlike containers, which export a unified interface, each of these systems is a separate extension to the Emulab model. Each scaling system determines the appropriate scaling parameters through feedback. A representative scenario is repeated using different parameters to estimate the proper scaling factors, e.g., the number of virtual machines per computer. The container system provides a priori guidance.

We discuss a few scaling technologies for network experimentation that are ripe for containerization.

ModelNet[20] simulates network topologies inside a few computers using virtual networking. It provides many thousands of packets per second of emulated network traffic through topologies and conditions specified by a user. Only a few computers and a high capacity interconnection are necessary. It is a powerful system ripe for combination with others using the container model.

Emulab offers path emulation[17], which algorithmically extends the emulation of conditions on network links to the emulation of conditions on paths. This facility is dynamically configured based on measured network conditions in FlexLab[16]. Path emulation offers a scalability advantage, and FlexLab offers arguably more realistic configurations of the same scaling mechanism. These are both attractive for containerization.

An emerging standard interface for configuring virtual machine monitors (VMM), libvirt[5] is becoming more prevalent. It simplifies the selection of a virtualization technology and its configuration. However, libvirt does not provide the abstraction and scaling descriptions that containers do, nor does it address networking and nesting as cleanly or completely as containers. Libvirt does offer the opportunity to simply incorporate many underlying VMM technologies into the container system. As our prototypes mature we expect to adopt libvirt where possible.

Emulab configures and monitors at least two levels of virtualization in its experimental apparatus, but researchers will need to configure concrete elements at various levels of abstraction. The Plush[1] system provides unified abstractions for configuring systems of heterogeneous computers. While it is not directly applicable for configuring concrete elements from disparate containers, we are likely to adopt similar abstractions.

On another axis, Stork[6] takes advantage of the physical collocation of multiple VMMs to efficiently configure them. Though not all containers can take advantage of the techniques Stork uses, we expect to use similar approaches to make them efficient. Our prototype makes some optimizations for configuration of virtual machines.

4 Prototype Implementation and Demonstration

To test our ideas we prototyped key aspects of the system and created several large-scale apparatuses. We wanted to show that the model will support interesting forms of virtualization, that the interconnection model is useful, and that containers can be implemented efficiently.

4.1 Virtualization

Our prototype makes several abstractions available to researchers: dedicated computer elements, virtual machine instances, and networked processes. If we consider fidelity as how closely an element represents a general purpose computer, the computer container makes whole computers available as high fidelity, high cost elements. Virtual machines are lower fidelity but lower cost. The process elements are very low fidelity, very low cost elements.

The process container is an unusual abstraction. That container puts each concrete element into a separate address space bound to a unique networking stack. We use the ViewOS[10] virtualization system to implement this abstraction.

The computer container is implemented by calling out to the DETER testbed interfaces, and the virtual machine container uses the QEMU[4] virtualization system. We can create virtual machine containers inside computer containers and process containers inside either computer containers or virtual machine containers. The implementation of these containers is straightforward, though we do apply some optimizations described in Section 4.3.

The prototype can create a nested structure of all three element abstractions (processes inside VMs inside computers). Containers are easy to compose across multiple layers of abstraction, which argues in favor of the generality of our design. Such generality implies that the system can interconnect larger structures as well, interconnecting multi-computer testbeds as it interconnects virtual machine monitors.

Each of our container implementations can parcel its resources into concrete elements or inner containers based on the descriptions mentioned in Section 2.1.

4.2 Communication

Our interconnection model is simple and powerful enough to produce complex topologies, to reuse standard connectivity tools at multiple levels, and to use as a base for future enhancements. We provide patch-panels at each container level that both interconnect and translate cross-container communications. This standard placement of a simple abstraction allows us to import a standard technology, Virtual Distributed Ethernet Switches (VDE)[7]. These switches and topologies can be enhanced to provide network emulations such as induced delay or loss models.

Our prototype implements communications both within a container and across container boundaries. Both are necessary to establish the complex connections that large scale network apparatuses require. The cross-container communication enables interaction across different representations with different fidelity.

We use an existing communication system to interconnect elements within the container, and container code to implement the cross container connections. Our prototype makes widespread use of VDE switches, because of their clean interface and interoperablility. Many VMMs that do not directly support VDE can be connected using tap interfaces.

The container code configures VDE switches to act as patch panels between concrete elements within a container, as well as connecting them outside containers through appropriate interfaces.

The container container provides the communication scope between containers by housing the VDE switch that connects the inner container's tap interfaces. Our recursive design made reusing VDE switches at multiple levels straightforward.

4.3 Scaling Techniques

Though we generally prefer to take advantage of the ongoing research in making virtualization more scalable, as a practical matter our prototype makes some optimizations to scale to an interesting size. We discuss our scalable file system deployment as one example of the sort of optimization that containers support.

Configuring many virtual machines can scale badly, especially when creating their file systems. Naively installing or replicating the same disk image for many instances of the same virtual disk will be wasteful. However, each machine will make small changes to the file system, so they cannot simply share an image. We used a copy-on-write system to allow sharing the a file system image between QEMU instances while preserving isolation.

All concrete elements share a common view of several file systems exported by the host testbed. Among other things, the physical machines share `/home` which is mounted via NFS. The file systems are exported to QEMU virtual machines using 9P [15], the file system from Plan 9. Processes share a file system with their host, so they have access to the same files whether they are embedded in physical machine or virtual machine.

We chose 9P because it is a network file system without many of the pitfalls of NFS or the complexity of AFS. The server is a cross-platform user-level daemon. It allows re-exporting NFS-mounted directories which allowed us to seamlessly make use of DETER nodes as hosts for virtual machines.

5 Demonstration

To exercise our prototype and test our ideas we created a large-scale apparatus based on the example we discussed in the introduction. We extracted a topology from the rocketfuel Internet routing topology database[19] and used that as an environment in which to release a DDoS attack carried by a bit of self propagating code. The code propagates according to a simple contagion model and each infected machine begins launching a few well-formed packets toward the designated victim. The code continues infecting new machines as the DDoS grows. The hypothetical researcher is studying the attack traffic and its effectiveness at the defender, varying the worm propagation or network topology.

We are able to scale this experiment by abstracting the elements and matching them to containers in the system. The Internet routing topology and infectable hosts are all very simple and abstracted as processes. The worm spread is modeled rather than executing exploit code on running web servers because the researcher is not interested in that aspect of the scenario. The destination host is a computer running a commercial operating system and the researcher may install experimental tools there to monitor the attack in detail. The DDoS command and control systems, all simple IRC based controllers, are realized as virtual machines.

Because the purpose of this demonstration was to test containers, we focused on creating a case that required multiple levels of abstraction and evaluating how well we could create and manipulate that. We do not claim that this is a novel or realistic propagation/DDoS scenario.

5.1 Building The Apparatus

We used custom tools to extract the routing topology from the rocketfuel database and output a topdl description of that topology. We manually matched the elements to their containers based on their abstraction level discussed above, and then manually grouped the process-level concrete components into containers. Eventually this step will be carried out by tools, designed along the lines discussed in Section 2.3. From here our prototype scripts configured the container containers that configured the virtual machine containers; the virtual machine containers configured the process containers to create the apparatus.

The largest demonstration topology we created consists of a total of 662 concrete elements created on 8

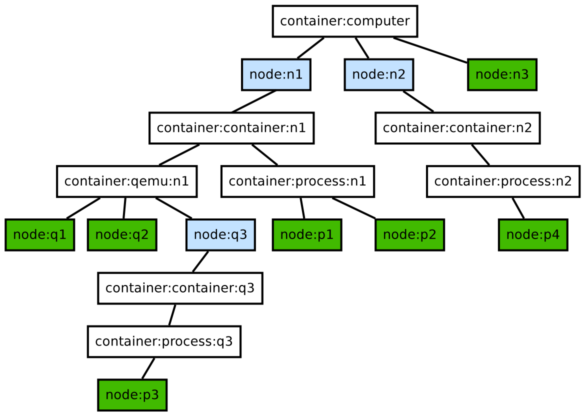

Figure 3: Embedding tree showing resources implementing containers and concrete elements

physical computers. We realized them as 1 physical machine, 4 QEMU virtual machines, and 657 lightweight processes. The lightweight processes are embedded within 26 additional QEMU virtual machines (making 30 total), which are in turn embedded within 7 physical machines. One physical computer is used directly as a concrete element.

In order to test our communication and nesting system, we embedded process containers inside virtual machine containers when the problem description did not strictly require that. Removing the extraneous VMs would yield higher performance and scaling.

Because our hypothetical researcher is primarily interested in the accumulation of attack traffic at the defender, the latency of multiple abstraction layers in the core is an acceptable price to pay for the high utilization. For reference, the overhead of crossing the abstraction boundaries made the longest path round trip time 25 ms for a 233 hop path.

Figure 3 shows a partial embedding. Light rectangles represent containers, lightly shaded ones represent resources allocated to containers, and dark rectangles those used as concrete elements. The labels consist of the type of usage (container or node), then for containers the type of container (computer, container, QEMU, process) and the resource hosting the container. Nodes are followed by their name.

The figure depicts the range of container nesting used in the prototype. It shows the DETER testbed (a computer container) that has allocated 3 computers, one directly as a concrete element and two as inner containers. One inner container, on node n2, contains a container container and a process container with a concrete element allocated in the process container. The other container includes a container container that holds both a QEMU virtual machine container and a process container. The process container holds a process element. The QEMU container holds two virtual machine elements and a process container that in turn holds a process element.

Each process represents an infected network, and represents multiple infected machines. We allowed those processes to represent as many as 1000 machines, meaning the entire apparatus models more than 50,000, though obviously at very coarse levels of abstraction.

5.2 Using the Apparatus

Even considering only the 662 concrete elements in the apparatus, Our ability to run standard experimentation tools was key. Running an experiment by logging into the elements or even scripting the action would be daunting with so many actors. SEER[18] was directly applicable.

SEER enabled coordinated use of the nodes and a real time visualization of the worm spread and the DDoS attack. The container-generated apparatus supported SEER operation with minimal changes, and the changes we did need have been communicated to the tool maintainers.

The result of this demonstration was a plausible, very large scale network apparatus that we could manipulate with existing tools. Because it was laid out along lines suggested by the abstractions of the problem, we were able to build that apparatus with limited intervention. Because the prototype uses resources aggressively, we were able to use very few physical resources to do so.

6 Conclusions

The container system addresses the need for large scale network research by using researcher's abstractions to guide the construction of large scale network apparatus. We have described the container abstraction that encapsulates a broad range of implementation techniques and characterizes their abstraction levels and scalability parameters.

We have shown our detailed system design for a selecting containers, organizing them into an abstract apparatus and creating a real apparatus from them. That real apparatus is conceptually compatible with experiment management tools and configuration systems that assist with carrying out large scale experiments.

Our prototype of the container model has demonstrated that the ideas have concrete, practical instantiations. We have been able to use standard tools like QEMU, VDE switches, and the DETER testbed to implement containers. The framework will easily admit new technologies, and we are specifically targeting using more VMMs using the libvirt interface.

We have demonstrated a large scale apparatus constructed using the container model, with some manual intervention. That system includes over 650 concrete elements using only 8 physical computers, and is capable of representing 50,000 computers at a coarse granularity in a specific scenario.

The demonstration apparatus can be controlled using conventional experiment management tools, though we expect future more scalable tools to be more appropriate for researchers.

Next steps include moving from the prototype to an operational service in DETER, making more kinds of containers available, and constructing a fully automated embedder. The embedder will remove the manual steps from apparatus creation.

Containers are a highly scalable, successfully prototyped, system for creating experimental apparatus being operationalized in the DETER testbed.

References

[1] ALBRECHT, J., BRAUD, R., DAO, D., TOPILSKI, N., TUTTLE, C., SNOEREN, A. C., AND VAHDAT, A. Remote control: Distributed application configuration, management, and visualization with plush. In *Twenty-first USENIX Large Installation System Administration Conference, LISA '07* (Nov. 2007).

[2] ANDERSON, L. P. T., CULLER, D., AND ROSCOE, T. A blueprint for introducing disruptive technology into the internet. In *HotNets-I '02* (Oct. 2002).

[3] BAVIER, A., FEAMSTERY, N., HUANG, M., PETERSON, L., AND REXFORD, J. In vini veritas: Realistic and controlled network experimentation. In *SIGCOMM '06* (2006).

[4] BELLARD, F. QEMU, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), ATEC '05, USENIX Association, pp. 41–46.

[5] BOLTE, M., SIEVERS, M., BIRKENHEUER, G., NIEHÖRSTER, O., AND BRINKMANN, A. Non-intrusive virtualization management using libvirt. In *Proceedings of the Conference on Design, Automation and Test in Europe* (3001 Leuven, Belgium, Belgium, 2010), DATE '10, European Design and Automation Association, pp. 574–579.

[6] CAPPOS, J., BAKER, S., PLICHTA, J., NYUGEN, D., HARDIES, J., BORGARD, M., JOHNSTON, J., AND HARTMAN, J. Stork: Package management for distributed vm environments. In *The 21st Large Installation System Administration Conference, LISA '07* (Nov. 2007).

[7] DAVOLI, R. Vde: virtual distributed ethernet. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on* (feb. 2005), pp. 213 – 220.

[8] DETER FEDERATION TEAM. Topology description language, Apr. 2011.

[9] FABER, T., AND WROCLAWSKI, J. A federated experiment environment for emulab-based testbeds. In *Proceedings of Tridentcom* (Washington, DC, 2009).

[10] GARDENGHI, L., GOLDWEBER, M., AND DAVOLI, R. Viewos: A new unifying approach against the global view assumption. In *ICCS (1)* (2008), M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5101 of *Lecture Notes in Computer Science*, Springer, pp. 287–296.

[11] GURUPRASAD, S., RICCI, R., AND LEPREAU, J. Integrated network experimentation using simulation and emulation. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on* (feb. 2005), pp. 204 – 212.

[12] HIBLER, M., RICCI, R., STOLLER, L., DUERIG, J., GURUPRASAD, S., STACK, T., WEBB, K., , AND LEPREAU, J. Large-scale virtualization in the emulab network testbed. In *In Proceedings of the 2008 USENIX Annual Technical Conference* (Boston, MA, June 2008), pp. 113–128.

[13] HIBLER, M., RICCI, R., STOLLER, L., DUERIG, J., GURUPRASAD, S., STACK, T., WEBB, K., AND LEPREAU, J. Feedback-directed virtualization techniques for scalable network experimentation. In *Flux Technical Note FTN200402* (May 2004), University of Utah.

[14] MIRKOVIC, J., BENZEL, T. V., FABER, T., BRADEN, R., WROCLAWSKI, J. T., AND SCHWAB, S. The deter project: Advancing the science of cyber security experimentation and test. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on* (nov. 2010), pp. 1 –7.

[15] PIKE, R., PRESOTTO, D., THOMPSON, K., AND TRICKEY, H. Plan 9 from bell labs. In *In Proceedings of the Summer 1990 UKUUG Conference* (1990), pp. 1–9.

[16] RICCI, R., DUERIG, J., SANAGA, P., GEBHARDT, D., HIBLER, M., ATKINSON, K., ZHANG, J., KASERA, S., AND LEPREAU, J. The flexlab approach to realistic evaluation of networked systems. In *In Proceedings of the Fourth USENIX Symposium on Networked Systems Design and Implementation (NSDI 2007)* (Cambridge, MA, Apr. 2007), pp. 201–214.

[17] SANAGA, P., DUERIG, J., RICCI, R., AND LEPREAU, J. Modeling and emulation of internet paths. In *In Proceedings of the Sixth USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, Apr. 2009), pp. 199–212.

[18] SCHWAB, S., WILSON, B., KO, C., AND HUSSAIN, A. SEER: a security experimentation environment for deter. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007* (Berkeley, CA, USA, 2007), USENIX Association, pp. 2–2.

[19] SPRING, N., MAHAJAN, R., AND WETHERALL, D. Measuring isp topologies with rocketfuel. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2002), SIGCOMM '02, ACM, pp. 133–145.

[20] VAHDAT, A., YOCUM, K., WALSH, K., MAHADEVAN, P., KOSTIĆ, D., CHASE, J., AND BECKER, D. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev. 36* (December 2002), 271–284.

[21] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), USENIX Association, pp. 255–270.