EasyScale: Easy Mapping for Large-Scale Network Security Experiments

Wei-Min Yao, Sonia Fahmy, Jiahong Zhu
Department of Computer Science
Purdue University
Email: {wmyao, fahmy, zhu206}@cs.purdue.edu

***Abstract*—Network emulation enables network security evaluation using unmodified implementations. Experimental *fidelity* with emulation is higher than simulation through the integration of real hardware and systems, but the *scalability* of emulation testbeds is limited. Scaling techniques such as virtualization and real-time simulation enable larger scale experiments. Using scaling techniques for network security experiments can, however, require considerable expertise in order to avoid overloading resources. In this paper, we propose a new framework, *EasyScale*, that aims to bridge the current gap between emulation testbed users and large-scale security experiments possibly using multiple scaling techniques. Our results from distributed denial of service and worm attack experiments demonstrate that EasyScale can easily allocate testbed resources to the critical components in an experiment, lowering the barrier for testbed users to conduct high fidelity yet scalable network security experiments.**

I. Introduction

As the Internet adopts new paradigms such as cloud computing, the difficulty of conducting realistic experiments increases. Internet-scale security experimentation is especially challenging since it is typically infeasible to perform the security experiments directly on a production network. Additionally, many security attacks involve both the data and control planes. For instance, BGP session resets have been observed with Denial of Service (DoS) attacks [1]. During the slammer worm propagation, BGP withdrawals increased, possibly due to ARP messages caused by traffic to nonexistent addresses, or due to excessive data transmissions by infected hosts in each enterprise [2]. Researchers thus resort to hybrids of simulation, emulation, and testbed experiments, employing a variety of *scaling techniques* [3]–[5].

Selecting an experimental platform entails a delicate balance between *scalability* and *fidelity*. On one side of the spectrum, network simulators, such as ns-2 [6], employ abstract models to simulate large networks on one or a few machines. The simplification of hardware, operating systems, or lower layers of the network protocol stack can adversely impact the fidelity of experimental results [7]. Further, researchers often need to experiment with real devices. For instance, the Stuxnet/Duqu/Flame worm attacks [8] target specific equipment in a nuclear facility. On the other side of the spectrum, network emulation testbeds include commodity hardware devices such as routers, switches, and PCs. This allows experimenters to run their unmodified applications on real systems and interact with specialized network devices. While emulation can provide higher fidelity, its scalability is limited. Popular emulation testbeds such as Emulab [9] and DETER [10] include a set of physical machines, usually a few hundred, shared among a large number of users. The testbeds often allocate resources conservatively to ensure fidelity. A one-to-one mapping between hosts in an experimental topology and machines in the testbed severely constrains the size of experiments that can be conducted.

Consider the example of studying the impact of a large-scale Distributed Denial of Service (DDoS) attack utilizing a massive botnet. It is important to explore defenses against this attack using realistic scenarios, but it is undesirable to have attack packets sent across Internet links. As a result, researchers are forced to use a smaller *quarantined* testbed. The results of a non-representative experiment can be misleading and unexpected bugs may not be discovered until the Internet protocol or application is deployed onto an operational network, causing severe damage.

Resource multiplexing [3], [11] and real-time simulation [4] have been employed to increase experimental scale. For example, the DETER containers system [11] can support experiments that are two orders of magnitude larger than the testbed. There are fidelity tradeoffs involved, however. No scaling technique is suitable for mapping all types of Internet-scale experiments onto small or medium size testbeds. We argue that using *more than one scaling technique*, based on the experimental scenario and the experimenter's requirements, may be necessary. However, applying scaling techniques to large-scale network security experiments is especially challenging. Many of the attacks involved in the experiments are designed to deplete specific physical resources. A sophisticated mapping, which cannot be automated using systems available today, is required to avoid experimental artifacts. Manual mapping is intractable for large experiments.

In this paper, we propose *EasyScale*, a framework to *automatically map experimental topology nodes onto testbed resources according to user-specified fidelity requirements*. EasyScale has three primary goals:

(1) Extensibility to New Architectures: The experimental topology and systems to be mapped onto a testbed can be arbitrarily large and complex. New network architectures such as data centers, wireless sensor networks, and smart power grids, are being proposed and deployed. EasyScale is extensible to custom architectures via a divide-and-conquer approach. An input network topology is partitioned into several *sub-topologies* by the testbed user. Sub-topologies are

downscaled separately using algorithms designed to utilize particular scaling techniques. The results are integrated to yield the final mapping for the entire input network. Since a sub-topology is smaller and potentially less complex than the entire input network, its mapping can be easier. New network architectures can be supported by adding new sub-topology mapping algorithms into the system as *plug-ins.*

(2) User-specified Resource Allocation: Different components of a large-scale experimental topology are not equally important to the experimenter. Components that require high fidelity experimentation can vary depending on the goal of an experiment. To provide user-friendly resource allocation with maximum flexibility, EasyScale is designed to support both coarse-grained and fine-grained allocation schemes.

Using a fine-grained allocation scheme, testbed users can have full control over how virtual nodes are mapped to physical machines. Such mapping cannot be automated in systems available today which support only basic graph partitioning algorithms. In EasyScale, a single value, a *fidelity index*, is assigned to each component in the topology. More resources will be allocated to components with a high fidelity index whenever possible. Testbed users can easily apply existing or customized mapping algorithms to generate the desired mapping.

A coarse-grained allocation scheme partitions a large-scale network to multiple sub-topologies. Components within a sub-topology can have the same fidelity index which represents the importance of this sub-topology. Testbed users may utilize scaling techniques at different *scaling levels* (Table I) in different sub-topologies. Unlike systems available today that ask users to allocate testbed machines to sub-topologies manually, EasyScale provides an efficient algorithm to automate the allocation process. With EasyScale, both fine-grained and coarse-grained schemes can be combined to satisfy a wide diversity of resource allocation requirements.

(3) Automated Configuration: Once a mapping from the experimental topology to physical testbed resources is determined, establishing the experiment on the physical machines is time-consuming for the experimenter. The configuration process not only requires strong expertise in the underlying scaling techniques, but also involves a large number of configuration scripts. For example, more than 10,000 lines of configuration files are generated to configure the 439 node network topology in Section III on 45 DETER machines. Automating the configuration process is a key goal of EasyScale.

The remainder of this paper is structured as follows. Section II explains our proposed methodology. Section III discusses our experiments and results. Section IV summarizes related work. We conclude in Section V.

II. Methodology

A. Input and Output

From the experimenter's perspective, the input to EasyScale consists of three items: *testbed resources*, *virtual topology*, and *user specifications*.

The *testbed resources* input describes the resources available in the physical testbed. Similar to the setup in Emulab [12], homogeneous physical machines can be grouped together to form a *hardware type*. Resource limitations such as the capacity of physical links can be specified as attributes of the particular type. The testbed resource information can be obtained from the testbed administrator and users typically need only specify the number of machines they plan to use.

The *virtual topology* is the set of virtual nodes and links that forms the experimental topology. The topology information is represented using the format used by the underlying testbed, making EasyScale transparent to the experimenter.

By default, EasyScale maps the entire virtual topology to testbed machines using METIS [13] and scales the testbed with OS-level virtualization. For sophisticated mapping strategies, users are able to guide the EasyScale process with additional information that we label *user specifications*. This includes:

(1) Sub-topology information: Virtual nodes with similar properties can be grouped together by the experimenter to form *sub-topologies*. For each sub-topology, an associated scaling level is selected (e.g., from Table I). Fig. 1 depicts an example experimental topology partitioned into three sub-topologies, including two small LANs and one backbone network. The experimenter may elect to downscale the backbone networks using a lightweight OS-level virtualization technique, and to downscale the end-hosts via KVM virtual machines to support various guest OSes.

(2) Virtual node information: For each element in an experimental topology, the experimenter can specify a fidelity index, as well as the hardware type that the element should preferably be mapped to. The fidelity index is an integer, selected from an arbitrary range of integers, to indicate the relative importance of the element. Other detailed resource requirements such as memory or disk usage can also be specified (but are optional) and will be considered as additional constraints during the EasyScale mapping process.

TABLE I: Sub-topology scaling levels and associated scaling techniques.

Scaling Level	Scaling Technique	Examples
I	One-to-one mapping	DETER, Emulab
II	Full-hardware virtualization	KVM, VMware
III	OS-level virtualization	LXC, OpenVZ
IV	Real-time simulators	ns-3, PRIME
V	OS processes	View OS

The output of EasyScale includes a downscaled network topology and a set of configuration scripts. The downscaled topology can be loaded onto the physical testbed using a one-to-one mapping. The testbed is initialized by running the configuration scripts on each physical machine; launching virtual machines onto physical machines, for example.

B. The EasyScale Procedure

EasyScale is composed of three modules: *mapping generation*, *mapping selection*, and *testbed script generation*.

The *mapping generation* module is designed to generate a set of possible mappings from the input virtual topology to physical machines. The virtual topology is not assumed to be suitable for analyzing or downscaling using techniques such as [5], [14]–[16] which only focus on data plane operation (and some of which, e.g., [14], make assumptions about traffic models). Since the virtual topology can be arbitrarily large and complex, our first step is to partition the virtual topology into manageable pieces.

The virtual topology is partitioned according to the hardware type and sub-topology information associated with the virtual nodes. Virtual nodes that map to the same hardware type are processed together (in the mapping selection module described below) to generate a mapping to the physical machines of that hardware type in the testbed. Since most testbeds contain only a few sets of homogeneous physical machines, each sub-topology is mapped to a single hardware type in most cases.

After the partitioning, mapping algorithms (*downscale plug-ins*) are applied to each sub-topology to generate a set of mappings. Once all feasible mappings are generated by the *mapping generation* module, the *mapping selection* module selects the best combination of sub-topology mappings. Using the mapping results, the *testbed script generation* module produces the EasyScale outputs for a particular testbed. For example, all experiments in Section III utilize the module designed for the DETER testbed that leverages the Cloonix [17] software.

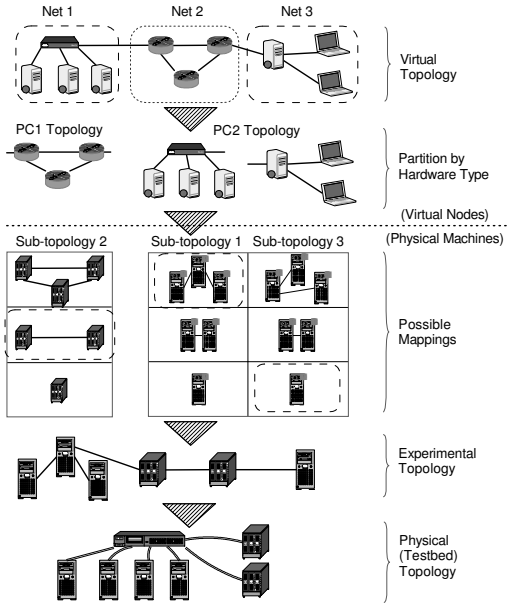

Fig. 1: Example mapping procedure.

The complete procedure is shown in Fig. 1. In this example, the virtual topology can be split into three sub-topologies (Net 1, Net 2, and Net 3). There are two types of physical machines in the testbed: two high-end servers (PC1) and four general servers (PC2). Virtual nodes in Net 2 should be mapped to PC1 machines and the remaining nodes to PC2 machines.

The virtual topology is first partitioned by hardware type. For instance, the PC2 topology in Fig. 1 denotes the part of the virtual topology to be mapped onto PC2 machines in the testbed. For each sub-topology and hardware type, multiple mappings are generated using the downscale plug-in. For example, the second mapping of sub-topology 3 can be generated by hosting the two laptops on one PC2 machine with virtual machines, and the server on another PC2 machine.

After possible mappings are generated, the optimal combination of mappings is selected by the *mapping selection* module using a dynamic programming algorithm. Selected mappings together represent the experimental topology that should be loaded onto the testbed. In this example, the ten virtual nodes are first mapped to the experimental topology formed by six testbed machines. Physical devices are configured by the testbed to provision the experimental network. Several virtual machines can then be launched on the machines in the experimental network to establish the desired network testing environment.

C. Mapping Generation Module

The most important question that must be addressed in the mapping generation module is how to select the most appropriate mapping algorithm for each sub-topology.

Based on the sub-topology scaling level specified by the experimenter, the associated *downscale plug-in* is applied to generate a set of possible mappings from the virtual nodes to testbed machines. A downscale plug-in is an implementation of a mapping algorithm that typically maps several virtual nodes to fewer physical nodes using a specific scaling technique. For example, a simple plug-in can be implemented using the METIS graph partitioning library [13]. The virtual topology can be directly partitioned by METIS and then virtual nodes in the same partition can be hosted using LXC virtual machines onto the same physical machine.

To give the user more control, EasyScale also supports a *mapping template* mechanism. Consider a user who wants to explicitly assign two virtual nodes onto a single physical machine, or a user who wants to allocate more resources to routers with higher degrees. The EasyScale mapping template (illustrated in Fig. 2) allows users to customize downscale plug-ins from a set of mapping rules. For instance, if a specific set of virtual nodes should be assigned to the same testbed machine, the template will pre-process the input topology and merge these nodes into a single node before invoking a graph partitioning library.

The mapping template also provides the ability to ensure that the generated mappings do not overload physical links. Fig. 3 gives an example on how the mapping template can be configured to take advantage of existing graph partitioning algorithms. In this example, there are 100 networks generated using RocketFuel [18] where each network contains 200 routers and all links are 100 Mbps. We partition each network to various numbers of partitions using four different downscale plug-ins. A partitioning result is considered a valid mapping if the aggregated virtual link bandwidth between any two partitions does not exceed the physical link bandwidth between two physical machines (1 Gbps).

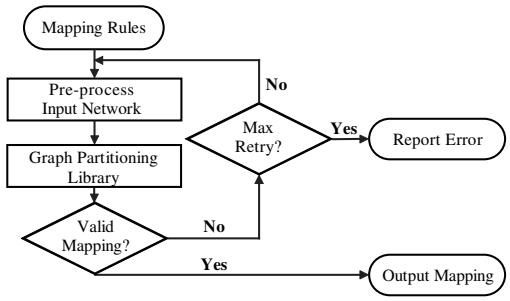

Fig. 2: The mapping template.

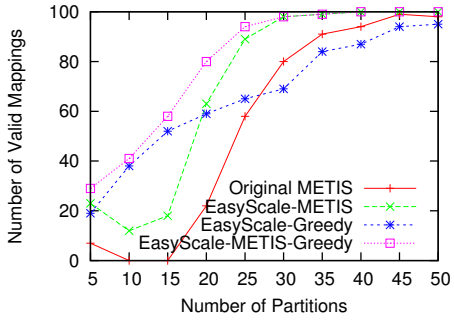

Fig. 3: The number of valid mappings generated by different mapping algorithms for 100 test networks.

As seen in Fig. 3, most mappings generated using the original METIS graph partitioning library failed to satisfy the physical link bandwidth restriction. With the mapping template, one can easily integrate existing graph partitioning software to customize the downscale plug-in. For example, the EasyScale-METIS plug-in is configured to automatically retry up to 10 different METIS mappings to increase the likelihood of finding a valid mapping. The EasyScale-Greedy plug-in takes advantage of a greedy graph partitioning algorithm that tries to minimize the the number of virtual links between partitions. The results demonstrate that this plug-in is more effective in generating mappings with a small number of partitions. To benefit from both algorithms, we can configure the mapping template to apply the greedy algorithm after 10 invalid METIS mappings. As expected, this hybrid plug-in is able to produce the most valid mappings.

D. Mapping Selection Module

The goal of the mapping selection module is to select the best combination of sub-topology mappings that requires no more than the available testbed machines. For example, in Fig. 1, the two highlighted mappings are selected from the sub-topology mappings of sub-topology 1 and 3. The two selected mappings together form the mapping for the general server (PC2) machines in the testbed. Two questions must be addressed by the *mapping selection* module: (1) How to compare sub-topology mappings, and (2) How to efficiently select the best combination among all sub-topology mappings.

Assume the hardware type is *nodeType*, and the virtual topology is partitioned into *numParts* sub-topologies. Let $vTop$ be the set of nodes in the virtual topology assigned to the *nodeType* machines by the experimenter. Let $vTop_i$ be a subset of $vTop$ which contains only the nodes in the ith sub-topology, and *numMachines* be the number of available *nodeType* machines in the testbed.

1) Evaluating Sub-topology Mappings: A *value function* is used to compare sub-topology mappings, i.e., a sub-topology mapping with a higher value is more likely to be selected as output. In our implementation, two sample value functions are defined to balance the resource allocation among sub-topologies. The two functions are designed based on the following intuition: (1) the number of physical machines that a sub-topology is mapped to can be proportional to the number of virtual nodes in the sub-topology, or (2) it can be proportional to the aggregated fidelity indices of all virtual nodes in the sub-topology.

More formally, let $sumFI$ be a function that returns the sum of the fidelity indices of a set of virtual nodes. Given a sub-topology, $vTop_i$, the target number of physical machines, $targetSize_i$, to map $vTop_i$ onto can be calculated by equation (1), or by equation (2).

$$\frac{targetSize_i}{numMachines} = \frac{|vTop_i|}{|vTop|} \tag{1}$$

$$\frac{targetSize_i}{numMachines} = \frac{sumFI(vTop_i)}{sumFI(vTop)} \tag{2}$$

Given a sub-topology ($vTop_i$) and its mapping (map) onto physical machines, the value function, $valFn$, is defined based on equation (1) or equation (2) as:

$$valFn(map) = 1 - |\frac{targetSize_i - |map(vTop_i)|}{numMachines}| \tag{3}$$

When this fails to generate a mapping of a specified size, possibly due to resource constraints, a constant negative value will be assigned.

2) Selecting Sub-topology Mappings: For each sub-topology, *numMachines* sub-topology mappings are generated to represent all possible mappings from the sub-topology to the *numMachines* machines in the testbed. Consider a virtual topology with *numParts* sub-topologies. A brute force solution that examines all combinations of sub-topology mappings has $O(numParts^{numMachines})$ time complexity. Fortunately, the mapping selection problem can be solved with an efficient algorithm that utilizes the dynamic programming technique.

Among the $numParts \cdot numMachines$ sub-topology mappings, let map_{ij} denote the mapping that maps the ith sub-topology to j testbed machines. The value of map_{ij} is defined by equation (4):

$$value(i,j) = \begin{cases} valFn(map_{ij}) & \text{if } 1 \le i \le numParts \\ & \text{and } 1 \le j \le numMachines \\ -\infty & \text{otherwise} \end{cases} \tag{4}$$

We define the mapping selection problem as selecting a combination of *numParts* sub-topology mappings that has the highest sum of mapping values and requires no more than *numMachines* testbed machines.

Let $OPT[i, j]$ denote the sum of mapping values in the optimal mapping combination that considers only the first i sub-topologies and uses no more than j physical machines. The optimal substructure of the mapping selection problem gives the recursive formula in equation (5):

$$OPT[i,j] = \begin{cases} \max\limits_{1<x<j} \left(OPT[i-1, j-x] + value(i,x)\right) & \text{if } i > 1 \\ value(1,j) & \text{if } i = 1 \end{cases} \quad (5)$$

Based on equation (5), we can easily compute the *OPT* array in $O(numParts \cdot numMachines^2)$ time and retrieve the optimal mapping combination in linear time.

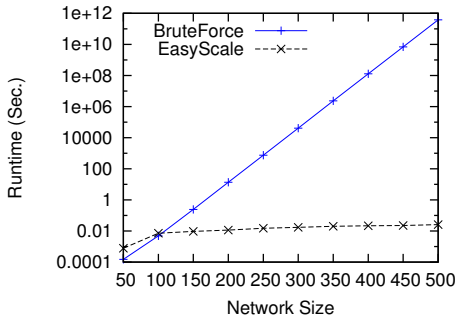

Fig. 4: Runtime of the mapping selection process using different algorithms. The brute force runtime is estimated using the first 10 M combinations.

Fig. 4 gives an example of the runtime comparison between the brute force and EasyScale solutions. In this example, there are 50 machines in the testbed, and the input network is composed of multiple 50 node sub-topologies. Since the sizes of the sub-topologies are small, the time to downscale all sub-topologies can be ignored. For example, both downscale plug-ins implemented in Section III-B take less than 30 seconds to generate all mappings for a 50 node sub-topology. As seen in Fig. 4, EasyScale scales well, whereas the brute force solution becomes intractable even with a medium-sized topology.

III. Evaluation

EasyScale is implemented in Python and contains a customized testbed plug-in to support the popular DETER testbed.[1] We evaluate EasyScale on two sets of experiments: (1) the impact of worm propagation on routing, and (2) distributed denial of service attacks.

A. Worm Attacks

The performance of Border Gateway Protocol (BGP) routing under stress was unknown until the Code-Red/Nimda worm attacks in 2001. A measurement study indicated that the attack was closely correlated in time with a large spike in the number of BGP routing updates [19]. Unfortunately, the measurement study was based on indirect observation from a set of monitoring points in the Internet and the causes of the spikes can only be inferred. In this section, we reproduce a worm attack scenario on the DETER testbed to study the BGP behavior directly and investigate two important questions: (1) What are the minimum testbed resources required to conduct an experiment? and (2) Will different resource allocation strategies impact the fidelity of the experiment?

1) Experimental Setup: We extract 500 connected ASes from the UCLA Internet Topology Collection project [20] to construct the network topology. The dataset was collected in January 2004, which is the earliest date available in the database; therefore closest to the actual date of the worm attack in 2001. Each selected AS is represented by a BGP border router emulated using the Quagga routing suite [21] on Linux. For simplicity, each BGP router is connected to its neighbors via a peering relationship. A randomly selected unique IP subnet is assigned to each router. By simulating the worm behavior, we are able to directly observe the BGP behavior under attack from a global point of view and confirm the correlation between the worm attack and the increase in BGP update messages.

The 500-node AS-level Internet topology is mapped onto the DETER testbed using two different mapping mechanisms. Since the network topology contains only BGP routers, all virtual nodes are grouped into a single sub-topology with scaling level III in Table I, where a virtual node is virtualized using Linux containers (LXC).

A baseline mapping mechanism simply utilizes the METIS graph partitioning library to distribute the LXC virtual machines evenly onto the physical machines, i.e., all nodes have the same fidelity index. We compare this baseline with an EasyScale resource allocation where the fidelity index of a node is assigned in proportion to the degree of the AS, i.e., more resources should be allocated to high degree routers, since they need to process more traffic and are more likely to become overloaded.

2) Resource Utilization: An experimenter typically wishes to use as few testbed machines as possible while ensuring the fidelity of the results. In this experiment, we monitor the CPU and memory usage to identify the minimum number of DETER testbed machines required.

Fig. 5 and Fig. 6 illustrate the maximum resource usage on all testbed machines during the experiment. From Fig. 5, the load of the network is light when no worm exists. Both mapping mechanisms can achieve satisfactory results with as few as 10 physical machines. However, during the worm propagation period, the load on the system increases significantly and more testbed machines are required to support the experiment. Fig. 6 shows that 50 testbed machines may still be insufficient to prevent overloading any physical machine in the baseline experiment. Fortunately, the number of testbed machines can be reduced if we allocate resources carefully. For example, only 20 machines are required with EasyScale degree-based fidelity indices.

[1] EasyScale is available at http://goo.gl/QD2aR. With a few modifications, the implementation can support most Emulab-based testbeds.

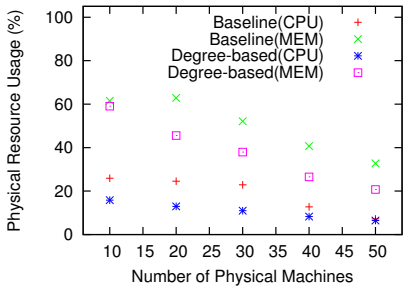

Fig. 5: Maximum resource usage without the worm using different number of physical machines.

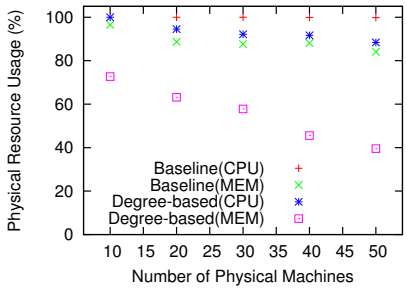

Fig. 6: Maximum resource usage under worm attack using different numbers of physical machines.

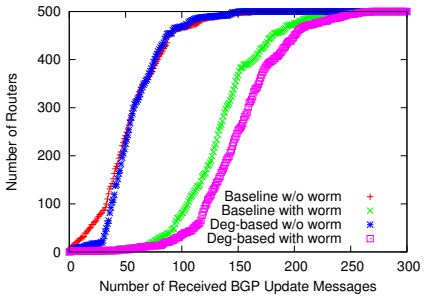

Fig. 7: The number of BGP update messages received by each router.

3) Resource Allocation vs. Experimental Fidelity: We now use 50 testbed machines to investigate the impact of resource allocation on the experimental results. The volume of BGP update messages received by each router is shown in Fig. 7. The results from both mapping approaches are consistent with the literature [19] in the way that the volume of BGP update messages increases under a worm attack. With degree-based EasyScale, we observe additional BGP messages received by the routers, giving stronger evidence of the worm attack.

A closer look at the two experiments reveals a more balanced resource utilization in the EasyScale experiment. The CPU utilization of the 50 testbed machines ranges from 2% to 97% in the baseline experiment and from 32% to 84% in the degree-based experiment. Several routers hosted on the machines with high CPU usage stopped processing routing messages during the attack. This results in fewer BGP messages in the baseline experiment. Thus, incorrect resource allocation can introduce artifacts to the experimental results.

B. DDoS Attacks

Distributed Denial of Service (DDoS) attacks still pose a significant threat to today's networks. In this section, we showcase how a large-scale DDoS scenario can be evaluated in an emulation testbed with the help of EasyScale. The attack scenario is designed based on a one-year web trace collected from a busy Purdue web server. The goal of the experiment is to understand the availability of our web server to visitors during the attack. The top 200 domains of our visitors, which cover more than 70% of the service providers of all visitors, are selected to represent the legitimate users. An additional 50 domains – that represent the DoS attackers – are selected from the black list generated by DShield.org. The legitimate web requests and the DoS attack traffic are sent from the 200 visitor domains and the 50 attacker domains to the web server.

Each domain is represented by a single virtual node. The network topology is then created by traceroute logs to the 250 domains. We perform basic processing to reduce the size of the topology. For example, we aggregate the last few hops in a traceroute record if they do not appear in any other traceroute record. After such reductions, the size of the network topology drops from 1232 to 439 nodes. The link bandwidth and the legitimate traffic are carefully chosen so that the system achieves high utilization but is not overloaded without the attack. All links in the topology are set to 10 Mbps and all routers are linked using the BGP routing protocol with peering relationships. During the experiment, 50 web requests are generated from each user domain and a full speed UDP flood is sent from each attack domain. The interval between requests is exponentially distributed with the mean set to 20 seconds.

1) Experimental Setup: We use 45 machines on the DETER testbed with dual 3 GHz Intel Xeon processors and 2 GB of RAM. Given the fixed physical resources, the experimenter needs to obtain the results that have the highest experimental fidelity. A sophisticated resource mapping from the virtual nodes to the physical machines is hard to predict before conducting the experiment. Therefore, we compare two sets of experiments with different resource allocation strategies.

In a baseline experiment, we simply assume all nodes in the experimental topology are identical and allocate the physical resources evenly using the METIS graph partitioning library. The METIS mapping is generated by partitioning the virtual topology into 45 pieces using METIS, with the goal of minimizing the edge cut between partitions.

We compare the baseline with EasyScale. We assign different fidelity indices to virtual nodes according to their node degree and their roles in the experiment as listed in Table II. Routers with higher degrees need to process more attack traffic, and the web server is the most important component in the network. Since the DoS attack focuses on network resources, we would like to map the end hosts (web users and DoS attackers) and the routers onto different DETER machines to avoid overloading the routers and the emulated network links. In EasyScale, this can be achieved by grouping virtual nodes into three different sub-topologies, namely, the 200 web users, the 50 DoS attackers, and the web server and routers.

In the baseline experiment, only a single scaling technique, *LXC virtualization* (scaling level III in Table I), is used. All 439 virtual nodes are hosted as virtual machines using LXC with the OpenWrt image. In the EasyScale experiment, *LXC virtualization* is applied only to the web server and routers. For DETER machines that only have end hosts, *OS processes* (scaling level V in Table I) are utilized to avoid the overhead

of virtualization.

Two downscale plug-ins that implement specialized mapping algorithms (taking the testbed resource limitations into consideration) are selected to perform the mapping for the LXC and OS process techniques. The LXC plug-in reduces the size of an input topology by continuously selecting a pair of nodes and mapping them onto the same physical machine. The pair of nodes is selected greedily: two neighboring nodes with the smallest fidelity indices are selected if the total resource requirements of both nodes can be fulfilled by a single testbed machine. For example, the number of outgoing links should not exceed the number of network interface cards installed on a physical machine. The plug-in designed for OS processes operates similarly, but merges nodes that can be aggregated (i.e., end hosts that connect to same router).

The value function based on equation (2) is used by EasyScale to create a balanced resource allocation between multiple sub-topologies. The 45 DETER machines are automatically allocated by EasyScale as follows: 20 machines are assigned to the 200 web users, 5 assigned to the 50 DoS attackers, and 20 to the remaining 187 nodes.

TABLE II: Fidelity Index Assignment

Virtual Nodes	Routers (degree) (≤ 3)	(4 or 5)	(≥ 6)	End Hosts	Web Server
Fidelity Index	2	3	4	2	8
Number of virtual nodes	100	50	27	250	1

2) Resource Utilization: Fig. 8 shows the cumulative distribution function (CDF) of the CPU utilization sampled during a DoS attack using 100-byte UDP attack packets. While most physical machines in the baseline experiment remain heavily loaded, we observed a significant drop in CPU utilization on 20 DETER machines in the EasyScale experiment. Since all routers are hosted on the 20 DETER machines, we can conclude that the EasyScale experiment provided sufficient resources, increasing fidelity.

3) Resource Allocation vs. Experimental Fidelity: To study the impact of DoS attacks on both the data and control planes of the network, we compute the success ratio of HTTP sessions and log the IP routing table on the web server. An HTTP session is considered successful if its response is successfully received by the client within 30 seconds [22]. The HTTP success ratio of the two experiments is depicted in Fig. 11 and Fig. 12.

The routing table on the web server is maintained by the BGP routing daemon. Although multiple IP addresses may be assigned to a virtual node, a single IP address from a unique subnet is selected to represent this node. Without aggregating the routing prefixes, each routing entry on the web server represents a known path from the web server to a virtual node. The routing dynamics are shown in Fig. 9 and Fig. 10.

In Fig. 9, we observed that a UDP flood attack using 1500-byte packets has no impact on the BGP routing in the EasyScale experiment. However, when the number of attack packets increases in Fig. 10, there is a direct impact on the BGP routing protocol. Since Fig. 8 indicates that we have allocated sufficient computational resources to all routers in the EasyScale experiment, routes are withdrawn because BGP keep-alive messages are dropped (due to the DoS attack).

In Fig. 9, several nodes are not always reachable in the baseline experiment. This indicates that the routers are overloaded due to insufficient resources and the resource allocation of the baseline experiment is introducing artifacts to the results.

Intuitively, the UDP flood attack using 100-byte packets should be more effective. However, in the EasyScale experiment, we observed flows with high HTTP success ratio in Fig. 12 which contradicts this intuition. Recall that in Fig. 10, since the DoS attack with 100-byte attack packets induces drops of legitimate packets, including the BGP keep-alive messages (Fig. 10), the parts of the network that include most DoS attackers are disconnected. As a result, legitimate users from the remainder of the network have a high HTTP success ratio and continue to have access to the web server during the attack.

To confirm this, we analyzed the network topology carefully and discovered that 156 of the virtual nodes (including routers, DoS attackers, and 69 legitimate users) are located in the regions where there are few DoS attackers. This finding is consistent with Fig. 12 where the horizontal portion of the EasyScale curve indicates that 33% of the web users have access to the web server during most of the attack period.

4) Experimental Fidelity Loss for a Smaller Experiment: We now explore experimental fidelity loss. Because of the current capacity of the DETER testbed, the 439 node network topology cannot be swapped onto the testbed directly. As a result, we further reduce the size of the topology to be able to conduct an experiment that can serve as the ground truth. All routers of degree two and end hosts are removed from the topology. A degree-two router is simply replaced by a 10 Mbps link and an end host is represented as a process running on the router to which it connects. Only the web traffic is transmitted in the network. After the reduction, the downscaled topology behaves similarly to the original network without dynamic routing and the DoS attack. We consider this downscaled topology as the ground truth.

We vary the load on the system by sending web requests at different inter-request intervals. A comparison of the HTTP success ratio between the experiments is shown in Table III. The only bottleneck to all web users is the 10 Mbps link in front of the web server. Ideally, this link should be able to support 200 web clients requesting a 50 KB file when the average inter-request interval is larger than 8 seconds.

When the average web inter-request interval is 20 seconds, the system is lightly loaded and the results from both experiments are identical to the ground truth. However, as the system load increases, the fidelity of the virtualized network drops. The fidelity loss results from insufficient physical resources on the machine that hosts the web server. More specifically, the virtual switch, a user-level program that manages all connections between the LXC virtual machines, fails to process all packets in real time. The fidelity of the virtualized network can be improved when the physical resources are allocated more carefully. For example, when the average inter-request

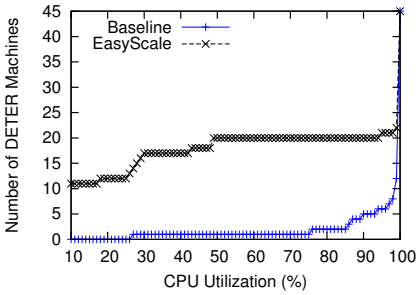

Fig. 8: CPU utilizations of DETER machines during a UDP flood using 100 B packets.

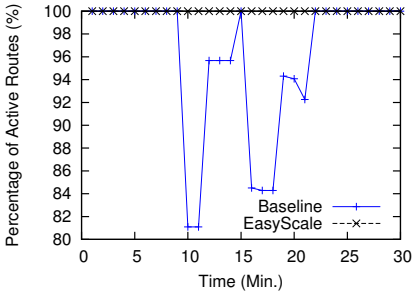

Fig. 9: Routing entries on the web server during a UDP flood using large packets.

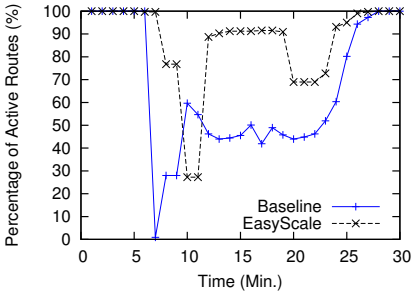

Fig. 10: Routing entries on the web server during a UDP flood using small packets.

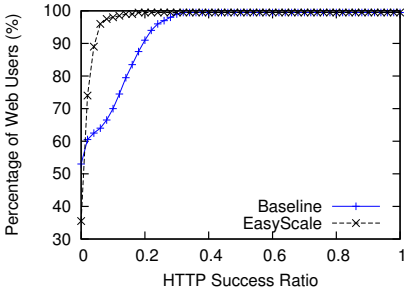

Fig. 11: HTTP success ratio during a UDP flood using 1500 B packets.

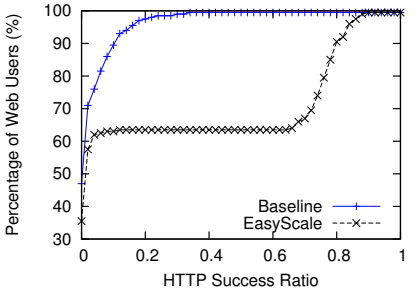

Fig. 12: HTTP success ratio during a UDP flood using 100 B packets.

Req. Interval	20	15	10
Ground Truth	100	100	100
Baseline	99.73	81.22	28.86
EasyScale	100	97.79	57.67

TABLE III: The HTTP success ratio collected from flows with different web inter-request intervals (seconds).

interval is 15 seconds, the fidelity of the EasyScale experiment is significantly higher than the baseline experiment.

Both experiments have low fidelity when the inter-request interval is shorter than 15 seconds. Physical machines must be added to avoid experimental artifacts in that case. Unfortunately, without conducting a ground truth experiment, it is often challenging to predict the best resource allocation strategy and identify the minimum physical resources required for a specific experiment. Monitoring the CPU utilization or router queues can aid in this prediction. One may need to examine multiple resource allocation strategies. With EasyScale, testbed users are able to quickly convert their resource mapping to a testbed experiment, which reduces the lengthy process of trial and error.

IV. Related Work

Testbeds scale by mapping multiple experimental resources onto available physical resources. Since the load of an experiment, such as link or CPU utilization, is typically unknown before conducting the experiment, finding a good mapping is non-trivial. An inappropriate mapping can overload physical resources and yield inaccurate experimental results [7]. As a result, although virtualization techniques are widely deployed in GENI [23] (including all its components such as Emulab and PlanetLab) and commercial cyber-ranges, most testbeds tend to allocate resources conservatively. Virtualization provides limited scaling capabilities for the routers and links in an experiment, and needs to be specified manually.

Real-time network simulators such as PRIME [4], ns-3 [24] and NCTUns [25] can interact with real network traffic. Integrating them with emulation testbeds, one can run unmodified applications on real machines and transmit packets to and from a simulated (and scalable) network. While real-time simulators combine the strengths of emulation and simulation, a major challenge is ensuring the computational power to allow the simulator to execute in real-time. Approaches such as DieCast [26] and SliceTime [27] have been proposed to execute experiments in slower virtual time to match the simulator speed. However, without running experiments in real-time, difficulties are encountered when trying to incorporate human actions or commercial hardware boxes into the experiment.

Container-based network emulators [28]–[30] enable high fidelity testbeds for small-scale experiments. With OS-level virtualization techniques, such as LXC, dozens of lightweight virtual machines with unmodified network applications can be interconnected and hosted on a single physical machine to construct the target network. However, the capacity of the emulator is constrained by the computation power, disk, and memory of the physical machine, and large-scale experiments cannot be supported.

The DETER containers system [11] is an infrastructure to support conducting large-scale experiments through virtualization. Unfortunately, only simple resource mapping mechanisms based on the METIS graph partitioning library [13] are implemented and users often need to generate the mapping from virtual nodes to physical machines manually for experiments

that require specific resource allocation strategies. With a simple script that converts EasyScale mapping results to network topologies, EasyScale can be seamlessly integrated with the containers system for more flexible mapping.

In addition to scaling the capacity of testbeds, several approaches have been proposed to pre-process the input experimental scenario to meet resource limitations. For example, DSCALEd [5] removes uncongested links in the network topology. The path emulator proposed by Sanaga *et al.* [31] simplifies a network path into a single hop using collected end-to-end observations from the Internet. Such reductions do not always provide precise guarantees on either the size or the fidelity of the reduced topology. Our FSP approach [16] partitions a large testbed experiment into a set of small experiments that do not exceed the testbed capacity. The set of experiments is *sequentially* executed in a few iterations to approximate the results of the original experiment. All these pre-processing approaches are orthogonal to *EasyScale* and scaling techniques. Combining methods from the two types of approaches can further increase the scale of testbed experiments.

V. Conclusions

In this paper, we have proposed a new framework, *EasyScale*, for easily configuring a large-scale network security experiment on an emulation testbed. Multiple scaling techniques, such as virtualization and real-time simulators, can be used for different parts of the input experimental topology in order to balance scalability and fidelity. Our resource allocation scheme considers user-specified fidelity requirements. While the default EasyScale configuration is sufficient for experiments with lightly loaded systems, sophisticated mapping for large-scale security experiments can be automated using sub-topology and virtual node information specified by the experimenter. Additional resources are allocated to the experiment components that are considered to be highly important, in order to increase the experimental fidelity.

With EasyScale, an inexperienced experimenter can easily run sophisticated large-scale experiments with little expertise in underlying scaling techniques. Results from our experiments with worm and DDoS attacks demonstrate the flexibility of EasyScale and its superiority over direct application of basic partitioning techniques such as METIS. EasyScale is extensible to a wide diversity of network experiments and emulation testbeds.

Acknowledgments. This work was funded in part by Northrop Grumman Information Systems, and by NSF grant CNS–0831353.

References

[1] Y. Zhang, Z. M. Mao, and J. Wang, "Low-rate TCP-targeted DoS attacks disrupts internet routing," in *Proc. of NDSS*, 2007.

[2] M. Lad, X. Zhao, B. Zhang, D. Massey, and L. Zhang, "Analysis of BGP update surge during Slammer worm attack," in *Proc. of IWDC*, 2003.

[3] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the Emulab network testbed," in *Proc. of USENIX*, 2008.

[4] J. Liu, Y. Li, and Y. He, "A large-scale real-time network simulation study using PRIME," in *Proceedings of Winter Simulation Conference (WSC)*, 2009.

[5] F. Papadopoulos, K. Psounis, and R. Govindan, "Performance preserving topological downscaling of Internet-like networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2313–2326, 2006.

[6] "ns-2," http://www.isi.edu/nsnam/ns/.

[7] R. Chertov and S. Fahmy, "Forwarding devices: From measurements to simulations," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 21, no. 2, pp. 1–23, 2011.

[8] "Israeli test on worm called crucial in Iran nuclear delay," http://www.nytimes.com/2011/01/16/world/middleeast/16stuxnet.html.

[9] "Emulab," http://www.emulab.net/.

[10] "DETER," http://www.isi.edu/deter/.

[11] DETER Team, "Building apparatus for multi-resolution networking experiment using containers," DeterLab, Tech. Rep. ISI-TR-683, 2011.

[12] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *ACM SIGCOMM CCR*, 2003.

[13] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[14] R. Pan, B. Prabhakar, K. Psounis, and D. Wischik, "SHRiNK: A method for enabling scaleable performance prediction and efficient network simulation," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 975–988, 2005.

[15] H. Kim, J. C. Hou, and H. Lim, "TranSim: Accelerating simulation of large-scale IP networks through preserving network invariants," *Computer Networks*, vol. 52, no. 15, pp. 2924–2946, 2008.

[16] W.-M. Yao and S. Fahmy, "Partitioning network testbed experiments," in *Proc. of ICDCS*, 2011.

[17] "Cloonix: Dynamical topology virtual networks," http://clownix.net.

[18] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," *ACM SIGCOMM CCR*, 2002.

[19] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Observation and analysis of bgp behavior under stress," in *Proc. of IMC*, 2002.

[20] "Internet Topology Collection," http://irl.cs.ucla.edu/topology/.

[21] "Quagga," http://www.nongnu.org/quagga/.

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

[23] "Global Environment for Network Innovations," http://www.geni.net.

[24] "ns-3," http://www.nsnam.org/.

[25] S.-Y. Wang and Y.-M. Huang, "NCTUns distributed network emulator," *Internet Journal ISSN*, vol. 4, no. 2, pp. 61–94, 2012.

[26] D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker, "DieCast: Testing distributed systems with an accurate scale model," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 2, pp. 1–48, 2011.

[27] E. Weingärtner, F. Schmidt, H. Vom Lehn, T. Heer, and K. Wehrle, "SliceTime: A platform for scalable and accurate network emulation," in *Proc. of NSDI*, 2011.

[28] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. of ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[29] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. of CoNEXT*, 2012.

[30] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim, "Core: A real-time network emulator," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, November 2008.

[31] P. Sanaga, J. Duerig, R. Ricci, and J. Lepreau, "Modeling and emulation of Internet paths," in *Proc. of NSDI*, 2009.