Beyond Disk Imaging for Preserving User State in Network Testbeds

Jelena Mirkovic, Abdulla Alwabel and Ted Faber
USC Information Sciences Institute
{*sunshine, alwabel, faber*}*@isi.edu*

Abstract

Many network testbeds today allow users to create their own disk images as a way of saving experimental state between allocations. We examine the effect of this practice on testbed operations. We find that disk imaging is very popular among both research and class users. Excessive disk image creation makes OS upgrades and patches time-demanding, leading over time to experiments that use old and vulnerable images. Since older images are not supported on new testbed hardware this hurts users by reducing their chance of successful resource allocation. Finally, disk images are usually large requiring excessive storage space on testbeds.

We then propose and evaluate three alternatives to disk imaging. We find that each approach significantly reduces storage requirements, and produces a list of OS image customizations that may help testbed users upgrade their images to newer OS versions. While this would still be a very manual process, we believe our results show promise and identify need for further research in this area.

1 Motivation

Network testbeds, such as Emulab [14], DeterLab [3] and Schooner [2], apply time-sharing of their resources between multiple users. This means that users acquire some amount of physical resources from the testbed for their exclusive use, and relinquish them when they anticipate they will not need them for a few hours. The testbed saves experiment definition consisting of node topology, node names, OS and node type choices, etc. so that a user can recreate the experiment later. Testbeds provide a set of *base OS images* that users can load on their experimental machines. An OS image is a block level image of the filesystem on a node, as described in [5]. Base images usually involve several Linux flavors, such as Fedora Core, Red Hat and CentOS, and some Windows flavors, such as Windows XP. Images contain some basic set of utilities and software that is needed by majority of users.

Often, users install additional software and OS modules during experimentation and may generate or download data on experimental nodes as well. This user-created state is removed from nodes when the experiment returns its resources to the testbed, and must either be explicitly saved by a user or recreated from scratch when the user resumes experimentation.

One way to recreate user state is to place this burden on users, who must set up desired state either manually or by scripting installation instructions and running them whenever they experiment. This is a significant burden for users, not only in the time needed to create scripts, but also in time and bandwidth cost added to experiment creation, if software to be installed and data to be placed on nodes must be fetched from remote sites.

Another possible solution is letting users hold on to the testbed resources until they complete experimentation. This is obviously undesirable for testbed operations, because it ties resources during user-idle times, when they could be used by others.

In fact, testbeds motivate users to release resources whenever possible, by measuring node idle time and reclaiming experiments that have been idle for several hours. Owners of long-running experiments that opt out from this policy are manually contacted by testbed operations staff to urge them to release resources if they do not need them. To help users preserve state they created, network testbeds enable users to create *custom OS images* of their nodes, store them on the testbed, and use them on a future resource allocation.

This paper is the first that investigates the impact of this policy on network testbed operation and on user experience. We obtain and analyze custom OS images and their use records from the DeterLab testbed [3]. Custom images offer significant utility to users, in that they capture changes made to the operating system, or at a very low level on the disks. We acknowledge these advantages, but note significant problems in disk image implementation and use practice.

We find that custom images continue to be used for an extended period of time after base image creation. As testbed software and hardware evolve, many of the base images used to create custom images become unsupported. This means that testbed staff does not patch these images and does not ensure that they support any new testbed functionalities, because staff time is limited. It also means that old base images are not ported to any new hardware, mostly because they lack driver support. Users that continue to use custom images created off of these old base images reduce their chances of successful resource allocation, because their experiments can only be hosted by a portion of the testbed's resources. This leads to dissatisfied users who do not understand that their difficulties are direct consequences of their actions, creates a bad reputation for the testbed and increases the number of support requests for the staff.

Another undesirable attribute of custom images in general, is that these are rarely patched for vulnerabilities. In case of testbeds that allow external traffic to experiments, like Emulab [14], this increases the chance of experimental machines being subverted by outside attackers and misused. While closed testbeds like DeterLab [3] do not run this risk, their staff occasionally discovers problems in base images and patches them. These patches do not propagate to custom images created off of these base images. For example, in March 2012 DeterLab staff has discovered and patched a race condition in its Ubuntu 10.04 image. The race condition made machines running this image unable to complete their boot process, and the entire resource allocation attempt would fail. The patch was advertised on DeterLab's Web page [3], clearly visible to all users. In three months, only 8 out of 32 custom images created from Ubuntu 10.04 base have been patched.

Yet another undesirable outcome of allowing users to create custom images is proliferation of these and significant storage they require. We find that almost half of testbed projects create custom images on DeterLab. In April 2012, these images occupied 340 GB of testbed storage — one third of the total shared disk space — often times leading to severe space shortage, affecting all testbed users.

We hypothesize that the main reason why users do not update or upgrade custom images is that they lack a detailed list of modifications they have made. Experimentation often occurs over weeks and months, evolving from an idea to a set of configurations, installations and scripts. Many customizations may be done manually over a long time period, and the image created this way may be used for months and years after. It would be very difficult for humans to recall the customizations they made and their order after such a long time.

In this paper we propose and evaluate three alternatives to disk imaging for preservation of user-created state:

1. ***DiffBase***: Storing files that are added or modified from the base image, and storing the list of files deleted from the image

2. ***DiffCustom***: *DiffBase* approach, extended to detect similar images within the same project. For this group of images, only one needs to be saved using *DiffBase* approach, while others can be preserved by storing differences between them and the first image.

3. ***AppStore***: Detecting common application installations within custom images and storing just the application configuration files. The rest of the image is stored using either *DiffBase* or *DiffCustom* approach.

Each of the proposed approaches would, to an extent, solve the image upgrade problem because it enumerates the customizations found in the image, making it easier for users to recall how and why they were made. It would further reduce storage requirements for images to only 19-25% of the current state.

2 Related Work

Other researchers have looked at tracking file system changes, though our goals of space saving and controlled updates are unusual. Of particular interest are disk imaging in the virtual machines such as VMware [13] or QEMU [15], and filesystem checkpointing, in the Emulab system [14], versioning filesystems, and filesystem checkpointing systems.

VMware [13], QEMU [15], and other virtual machine systems use filesystem level diffs to create snapshots. Each snapshot is a map of the blocks changed from a base image. These diff chains are similar to our proposed system in that they preserve differences between the base and the custom image, but these are preserved at the block level while we operate at the file level. Block-level diffs cannot be used to update or upgrade an image, and they do not provide a space-efficient upgrade path for new hardware, while file-level diffs can achieve all this. Further, block-level diffs can indicate larger than actual changes, if existing file's contents are merely shifted in the filesystem. Our system would record this as a file move, while the block-level system would regard it as a file change, resulting in a bigger state.

Emulab [14] has produced a system that allows users to checkpoint their ongoing experiments, including file system state [4, 10]. The system they use to capture filesystem state is conceptually similar to our method of capturing and overlaying filesystem diffs. They are concerned more with runtime speed than storage efficiency. Accordingly, they use copy-on-write-based branching to record differences from a base or ”golden” disk image at run time. The system modifies the Xen virtual machine to capture these changes at the disk block level [10].

Because we want to capture the semantics of modifications to the files in base disk image in order to simplify updates to the users' derived images, a block level approach is not attractive. We are also willing to pay the cost of off-line analysis to achieve better disk space savings.

Other filesystem implementations have captured update semantics at the file level, primarily to provide a fully versioned file system. The ext2cow system [8] began as an attempt to capture the evolution of the file system contents to ensure and enforce compliance with data retention and security regulations. It has become a general versioning system in which users can explicitly refer to file system contents on a particular date. The Elephant [12] file system tracks similar information, allowing the user to specify data retention policies per-file.

These systems provide new semantics – per-file-versioning across an entire filesystem – and are willing to pay for it in disk space. Each of them modifies the traditional file system semantics to write modifications to new blocks, and keep the old blocks unchanged and indexed to a version of the file. While we would take advantage of such versioned file systems if they were in use, we do not depend on collecting such data.

Several systems provide block-level snapshots of a mounted file system. These include FreeBSD [7], Plan 9 [9], WAFL [6], zFS [11] and others. These retain blocks in the snapshots, indexed to the snapshot points, and store modifications in new blocks. Unlike file-versioning file systems, information about which files have been changed is only indirectly retained. These systems are less concerned with saving space and would not produce useful information for users who wish to upgrade their custom images.

3 Disk Image Usage on DeterLab

We now provide some supporting evidence about the frequency and the characteristics of disk image use on the DeterLab testbed [3]. Specifically, we wanted to know how prevalent is use of disk images in experiments, who uses them, for how long, and what portion of the testbed supports each given image.

Unfortunately, information needed to answer these questions is not readily preserved by the Emulab software, and must be inferred by piecing together various data sources, and fixing inconsistencies. In our study we use two information sources:

1. **DB:** DeterLab's database holding information about all existing disk images at the present, such as their creation time, the owner and the OS type and version number of the base image used to create them. We note that the last two pieces of information are filled manually by users and thus may be incorrect. We attempt to correct them in Section 4. A user may delete an old image from the database when it is no longer needed. This action also deletes the image from the disk and we lose all information about its origin and contents.

2. **USE:** DeterLab's activity logs specifying topologies and testbed state for each attempted resource allocation. From these logs we can mine information about how often and how long images are used (mined from experiment topology data), and what portion of the testbed nodes supports any given image (mined from testbed state data). The USE data source contains information about use patterns for both the present and the deleted images.

We first classify all images into three classes: USEDEL, DBUSE and DBNOUSE. Figure 1 illustrates relationships of these image classes and their sizes. If an image has been created, used and then deleted it would not appear in our first data source but it would appear in the second. There are 483 such images (USEDEL). We can calculate their usage statistics and coverage, but we cannot match them with the base image nor do we know when they were created. In that case we will assume that such image was created at the time of its first use in the activity logs. If an image has been created and deleted but never used we will not see it in any of our sources. For images that have not been deleted we have complete information, whether they are used or not. There are 148 images that were created but never used (DBNOUSE), and 497 images that were created, used and are still present in the database (DBUSE). This subset of 645 images (DBUSE+DBNOUSE) will be analyzed further in Section 4. We analyze 1128 images that belong to the VISIBLE set (DBUSE+DBNOUSE+USEDEL) in this Section. In the Figure 1 the numbers in circles represent the count of images in each category that have never been used. Total of 148 out of 645 images present in the DB data source, or 22%, have never been used. Deleting just these would reduce the storage requirement to 78% of the current one.

We first investigate how prevalent is image creation. On Emulab-like testbeds such as DeterLab [3], Emulab [14], Schooner [2] and numerous others [1], senior researchers and teachers create *projects*, that junior re-

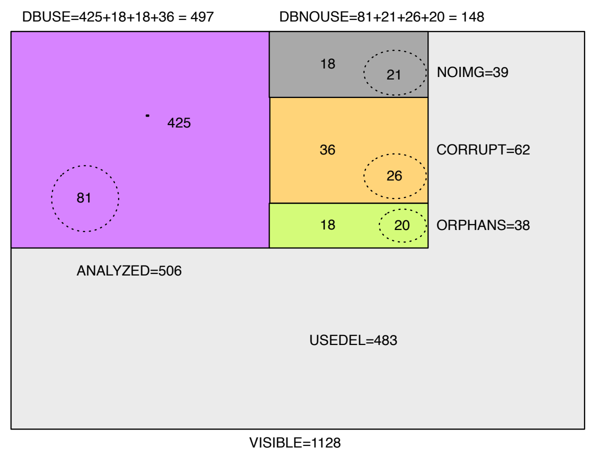

Figure 1: Image classes and their sizes. Numbers in circles denote count of unused images, while numbers outside the circles show counts of used images.

searchers and students join. Usually experiments and images can be shared easily within a project, but with great difficulty across projects. We thus analyze image use at the project granularity. We say that a project is *active* if it has ever requested testbed resources. An inactive project cannot create an image. On DeterLab there are 219 projects that were ever active. We classify them by their purpose into either research, class or internal (lead by DeterLab's operation staff). There are 161 research projects, 43 class projects and 11 internal projects. There were 4 projects we could not classify, because they were deleted. Out of all active projects, 104 have created those 1128 images, which is 47%.

We next wanted to associate each image with the project that created it, so we can analyze image creation activity per project. DeterLab's database holds information about image owners for those images that have not been deleted. To detect owners for deleted but used images (USEDEL), we use the fact that an image created by a project can only be used by this project. This access control rule is enforced by Emulab testbeds. The only exception to this are base images created by testbed staff that can be used by anyone. We then classify images used by multiple projects as base images and assign images used by only one project to that project. Out of 104 projects that created images, 77 were research projects (48% of all research projects), 19 were class projects (44% of all class projects), 6 were internal, and 2 we could not classify, because the projects and their descriptions were deleted from the database. Thus we can say that image creation is a popular activity among both research and class testbed users, with almost half of projects creating at least one image.

Research projects created total of 735 images, and class projects created 124 images. Internal projects created many more – 255 images, but most of these are created as base images for consumption by all users. We could not find an owner for 11 images. Figure 2 shows the number of images created per a research or a class project. While averages are 10 for research projects and 7 for class projects, detailed analysis reveals that image creation activity is heavily dominated by a few projects. We find that most projects create a few images — 21% create only 1 and 50% create less than 5 — but some create up to 52 images! More numerous the project images are, there is a higher chance that most of them are lying unused and that they are not being patched.

We next investigated how often disk images were used and for how long. There were 74,645 resource allocations on DeterLab between 2004 and 2012. Out of these 74,633 used a base image and 10,811 used a custom image. [1] Thus 14.4% of allocations use a custom image. If we calculate an age of an image from its base's creation date, we find that 12,231 allocations used an image from our ANALYZED set (see Figure 1 and Section 4 for explanation of this set) and 5,767 of those, or 47%, used an image older than one year. Older images are more likely to have vulnerabilities, and more likely to be unsupported by new testbed hardware.

We next investigated if custom image creation and use get more or less popular over time. Figure 3 shows the number of custom images created per year, indicating that image popularity is growing over time. To investigate use trends, we placed each custom image used in a year into a category based on what percentage of DeterLab's PCs that exist in a given year support this image. Figure 4 shows the number of custom images in use each year that are supported on less than 50% of DeterLab's

[1] It is possible for an allocation to use both base and custom images for different physical nodes.

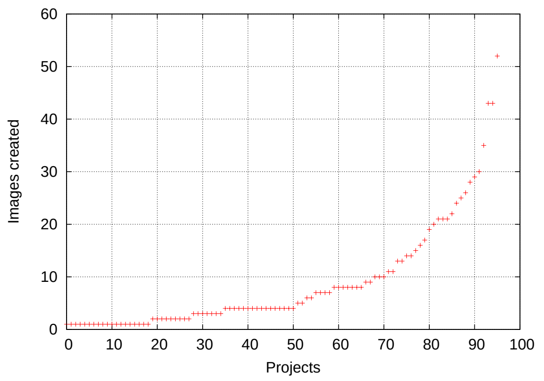

Figure 2: Images created per project (only for research and class projects that created at least one image).

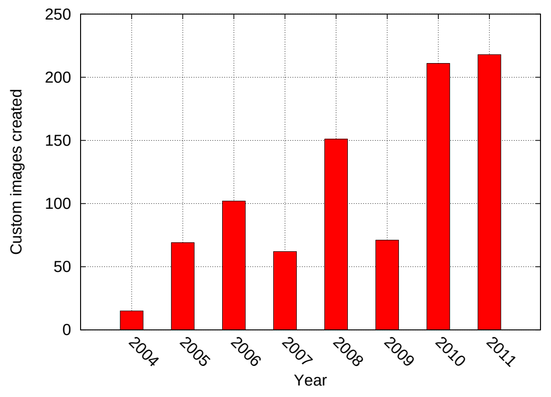

Figure 3: Custom images created per year.

PCs, on 50-80%, on 80-90% and on more than 90% of DeterLab's PCs. We note that the number of custom images in use is increasing. Further, there is a significant number of custom images in use each year that are only supported on a fraction of the testbed. For example, in 2011 there were 15 images supported on less than a half of the testbed PCs.

4 Alternatives to disk imaging

In this Section, we propose three alternatives to disk imaging for preserving user state in network testbeds. While our ideas are general enough to work on any network testbed, the implementation and measurement details we lay out here are specific to testbeds running Emulab [14] software.

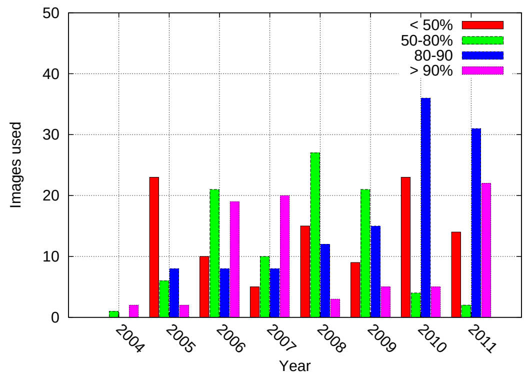

Figure 4: Custom images in use per year, broken down into categories based on what portion of DeterLab's PC pool supports them.

4.1 DiffBase

Our *DiffBase* approach compares a customized image, file by file, with its base during disk image creation, and stores the differences instead of storing the entire custom image. Files that are deleted from the base image are stored in a list. Files that are added to or modified from the base image are stored in their entirety. For sparse files, where most of the file is empty, we store the file size and the identifiers and contents only for those file blocks that contain data. Similarly, we detect and store information about hard and soft links that helps us recreate them on OS load.

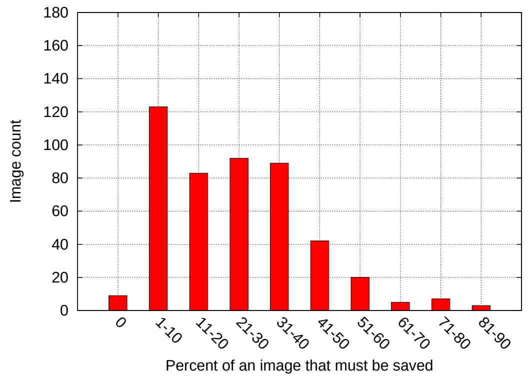

Figure 5: Storage needs when using *DiffBase* approach, compared the with current system, which saves the entire image.

To evaluate the savings of this approach we simulated the *DiffBase* approach on all the custom images on DeterLab. We started with the DB set (DBUSE+DBNOUSE), as defined in Section 3 and first attempted to locate and OS load each image. We discovered that 62 images were corrupted and would not load (CORRUPT set in Figure 1), 39 images were deleted from the file system but their records were present in the database (NOIMG set), and 38 images were orphans (ORPHAN set) – they claimed to be created from a base image that did not exist in the database at the time of our analysis. Removing all these classes left us with 506 images in the ANALYZED set. Among them were 33 base images and 473 custom images. All the image classes and their sizes are illustrated in Figure 1; dashed circles denote the number of images that were not used by anyone.

We first attempted to pair a custom image with its base, using the data recorded in DeterLab's database. However, this data is entered manually by users and may be incorrect, leading to large differences between the base and the custom image. We manually verified that this is a case for several images, by mounting them and identifying the OS type and version by executing `uname -a`, looking at the file system type, and checking `/etc` directory for release information. Often, this returned a different OS version, and sometimes even a different OS type than the one specified in the database. To overcome this issue we compared each custom image against all base images and paired them with the one that had the most similarity (and thus the smallest difference). This analysis occasionally associates two images that are not directly related, because both the base image and the custom image can be updated. If a base image is patched to remove a large software package, an image derived from the base (that still contains the package) may appear to have been derived from a different base image that contains that package. We have noted some such false correlations in our analysis, but they are uncommon. In real deployment, the Emulab software keeps information about the images loaded on all experimental nodes, and could accurately identify the base for each custom image that is being created, thus this issue would not arise.

Figure 5 shows on the y-axis the count of the custom images, for which only the percentage specified on the x-axis needs to be saved, using *DiffBase* approach. For example, there were nine images that were identical to the base images and need not be saved. Also, there were 123 images or 26% of all custom images that had less than 10% difference from a base image. Overall, applying *DiffBase* approach would reduce the size currently used for disk image storage on DeterLab from 340 GB to 87 GB, or to 25%.

We now describe how our custom images would be loaded onto experimental nodes, and evaluate potential delays that are introduced by our changes to the OS load process. Currently, the OS load first loads the MFS onto each experimental node, and then invokes Frisbee [5] to copy the desired custom or base image to the node from the testbed repository. The duration of MFS load is fixed, but the duration of copying the image over grows linearly with the image size. Since copying occurs over the network shared with other experiments, there is a potential for large variability if other experiments are loading images at the same time. The current system then boots the image on the experimental node. The duration of that operation depends on the image's complexity. In each of our alternatives to disk imaging, the new system would copy and load the base image onto the experimental node, restore preserved user state by copying some files and/or installing packages, and then reboot the node.

Figure 6 illustrates the OS load in the current and the proposed system. While all our solutions should lead to space saving, they might prolong the OS load process because of added state copy and reboot time.

We next evaluated the delay the *DiffBase* approach adds to the OS load time on selected 12 custom images, created from the Ubuntu 10.04 base image. We

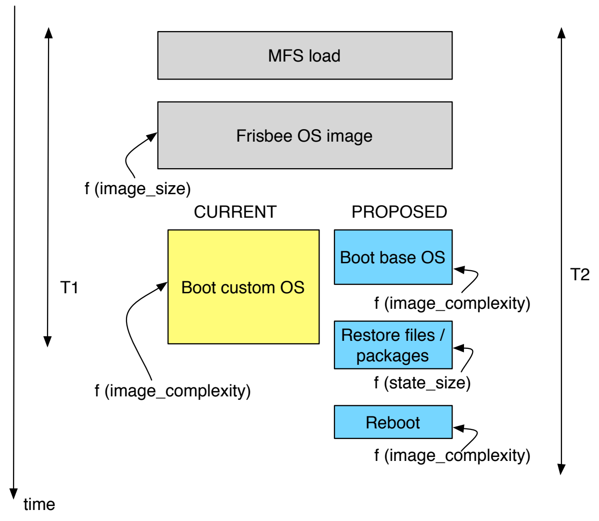

Figure 6: OS loading process in the current and the proposed system.

first loaded each custom OS on a separate experimental machine, and recorded the time to complete this operation – this is time *T1* in Figure 6. We took care to select the machines of the same hardware type, since we have noticed that the time to Frisbee an image over the testbed network depends on the hardware type. We then applied the *DiffBase* approach to identify and store differences between each custom image and its base. We next loaded the base OS on each of the 12 experimental machines, copied over a zip file containing the files that were saved using the *DiffBase* approach, unzipped them there and copied them to their proper directories. We further deleted the files that were identified for deletion by the *DiffBase* approach and restored soft and hard links, and sparse files. Finally, we rebooted each experimental node. We recorded each of the base-OS-load, restore and reboot times separately. Their sum represents time *T2* from Figure 6. This time will depend on the size of the base image (Frisbee copy), the size of the *DiffBase* state, and the complexity of the base image. Table 1 shows the images we tested ordered by their size, the sizes of *DiffBase* state, and the means and standard deviations for $T1$ and $T2$ extracted from 10 runs of the above experiment. Figure 7 shows the $T1$ and $T2$ times. All images booted successfully. For small images, time to load the image using *DiffBase* approach is comparable to the OS load time in the current system. For large images we tested (images 9-12) *DiffBase* approach surprisingly results in a *smaller* OS load time! This is because the size of each large custom image in our set is about 50% larger than the size of its base image plus the size of its *DiffBase* state. Since time to Frisbee an image dominates our measurements, a larger custom image leads to much larger OS load time in the current system.

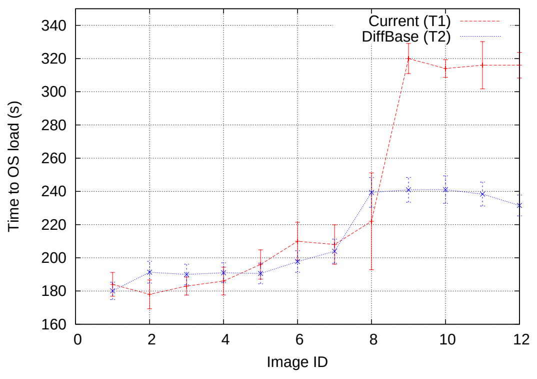

Figure 7: Time to OS load with the current and the *DiffBase* approach.

While *DiffBase* approach has its obvious advantages in space saving, there are certain limitations. Because *DiffBase* captures file contents, it cannot support research that requires saving state inside the file system but outside files. For example, studies of incompletely erased files or corrupted free lists would be compromised by the *DiffBase* approach. We note that the standard Emulab approach to disk imaging [5] also parses the file system to some extent and might compromise these studies.

4.2 DiffCustom

Our *DiffCustom* approach compares a custom image with another custom image from the same project, following our hypothesis that users create a series of related custom images as they explore options during their experimentation. For a group of similar custom images within the project, only one needs to be saved using *DiffBase* approach, while others can be preserved by storing differences between them and the first image.

To evaluate the savings of this approach, we simulated the *DiffCustom* approach on the ANALYZED dataset. For projects that create multiple custom images, we performed a pairwise image comparison within the project, and recorded the image pairs that are the most similar, as well as the size of the state that would need to be saved for them.

There were 124 custom images, where the owner project created at least one other custom image. For 16 of those, *DiffBase* generated a smaller state than *DiffCustom* indicating that they are branched off a different base image than the rest of their project's custom images. The remaining 108 images would benefit from the *DiffCustom* approach.

Figure 8 shows on the y-axis the count of the custom images, for which only the percentage of the image specified on the x-axis needs to be saved using *DiffCustom* approach. For example, there were ten images that need not be saved at all because they are identical to either a base image or another custom image. Also, there were 164 images, or 35% of custom images in our ANALYZED dataset, that had less than 10% difference from either a base image or another custom image. Overall, applying *DiffCustom* approach would reduce the size currently used for disk image storage on DeterLab from 340 GB to 75 GB, or to 22%.

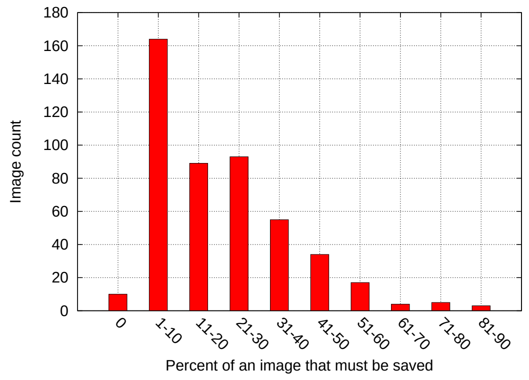

Figure 8: Storage needs when using *DiffCustom* approach, compared the with current system, which saves the entire image.

4.3 AppStore

While the *DiffBase* and *DiffCustom* approaches result in significant space savings, they produce large lists of files to be saved – commonly in hundreds or thousands. While such a list may help a user remember which customizations they performed on the base image, and thus facilitate upgrading an image to a newer OS version, this process would still be highly manual, tedious and lengthy. Our *AppStore* approach has a potential to address this shortcoming. This approach examines each file identified for saving by *DiffBase* approach, to detect if it belongs to an application installation we can recognize. Files that pass this check need not be saved by *DiffBase* approach, but can instead be installed from some central testbed repository upon each OS load. We would need to store the application configuration files though, so that we can preserve any customizations the user did to the application. The *AppStore* approach has a potential to both reduce the storage space requirement further, and to enable users to upgrade their custom images to a newer version of an operating system. The upgrade could be done by starting with a newer OS version, and by automatically installing packages identified by the *AppStore*. The remaining files, stored by *DiffBase* approach, would still need to be examined by a user, so that they could be upgraded or moved directly to the new image.

Image #	Imgsize (MB)	Diffsize (MB)	$T1_{mean}$ (s)	$T1_{stdev}$(s)	$T2_{mean}$ (s)	$T2_{stdev}$(s)
base	357		168	5.14		
1	357	55	184	7.21	180	5.23
2	357	55	178	8.66	191	6.52
3	358	55	183	5.4	190	6.19
4	360	200	186	8.36	191	5.97
5	420	63	196	8.81	190	6.22
6	465	100	210	11.39	198	6.52
7	489	135	208	11.9	204	7.25
8	516	267	222	29.2	240	8.94
9	1,034	465	320	9.08	241	7.48
10	1,035	465	314	5.28	241	8.29
11	1,034	465	316	14.22	238	7.20
12	1,035	465	316	7.73	231	6.27

Table 1: Times for restoring user state using *DiffBase* approach.

To estimate potential savings and applicability of the *AppStore* approach, we analyzed a sample of DeterLab's custom images containing 51 images created from Ubuntu 8.04 and 20 images created from Ubuntu 10.04. DeterLab currently mirrors repositories for these base images, although only selected packages are mirrored. For this selected set of custom images, we sampled file names from their *DiffBase* state. Sampling was done due to time constraints and it leads to underestimate of potential savings from the *AppStore* approach. Using the `apt-file` utility, we check each sampled file name to see if this file were part of a standard package. If so, we retrieve the list of all files in the package from the repository, and remove those files from the image's *DiffBase* state, if present.

Figure 9 illustrates potential savings by the *AppStore* method. It shows the count of custom images from the sample on the y-axis, for which only the percentage of their *DiffBase* state specified on the x-axis needs to be saved. We note significant savings for Ubuntu 10.04-based images and lower savings for Ubuntu 8.04-based images. This discrepancy occurs because DeterLab's local repository contains only a subset of available packages, and the size of this subset is much smaller for Ubuntu 8.04 than for Ubuntu 10.04. Overall, we project that only 70-80% of *DiffBase* state would need to be saved for those images that support package management. Thus, *AppStore* would potentially further reduce the size currently used for disk image storage on DeterLab from 340 GB to 65 GB, or to 19%.

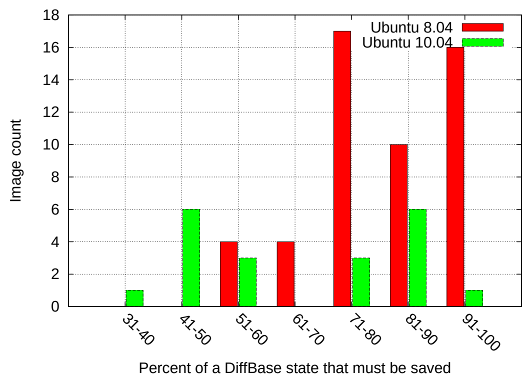

Figure 9: Storage needs when using *AppStore* approach, compared with the *DiffBase* approach.

5 Conclusions and Future Work

Disk imaging is extensively used in DeterLab, and likely in other network testbeds, for user state preservation. We find that such practice leads to extended use of old images that may have vulnerabilities, and are not supported on new testbed hardware. It also leads to significant use of sparse and shared storage space. We have proposed three alternatives to disk imaging. Each alternative would significantly reduce storage space requirements, and may help users upgrade their custom OS images to newer OS versions. While this paper offers strong evidence of space saving, we have only sketched potential solutions for custom image upgrade. Our future work will further investigate how to automate the upgrade process.

References

[1] Other Emulab Testbeds. `https://users.emulab.net/trac/emulab/wiki/OtherEmulabs`.

[2] Schooner WAIL (Wisconsin Advanced Internet

Laboratory).
http://www.schooner.wail.wisc.edu.

[3] R. Bajcsy, T. Benzel, et al. Cyber Defense Technology Networking and Evaluation. *Communications of the ACM*, 47(3):58–61, 2004.

[4] Anton Burtsev, Prashanth Radhakrishnan, Mike Hibler, and Jay Lepreau. Transparent checkpoints of closed distributed systems in Emulab. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 173–186, 2009.

[5] Mike Hibler, Leigh Stoller, Jay Lepreau, Robert Ricci, and Chad Barb. Fast Scalable Disk Imaging with Frisbee. In *Proceedings of the USENIX Annual Technical Conference*, 2003.

[6] Dave Hitz, James Lau, and Michael Malcolm. File system design for an NFS file server appliance. In *Proceedings of the USENIX Technical Conference*, 1994.

[7] Marshall Kirk McKusick and Gregory R. Ganger. Soft updates: a technique for eliminating most synchronous writes in the fast filesystem. In *Proceedings of the USENIX Annual Technical Conference*, 1999.

[8] Zachary Peterson and Randal Burns. Ext3cow: a time-shifting file system for regulatory compliance. *ACM Transactions on Storage*, 1(2):190–212, 2005.

[9] Rob Pike, David L. Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, and Howard Trickey. Plan 9. *Computing Systems*, 8(2):221–254, 1995.

[10] Prashanth Radhakrishnan. Stateful-Swapping in the Emulab Network Testbed. Master's thesis, School of Computing University of Utah, 2003.

[11] Ohad Rodeh and Avi Teperman. zFS: A Scalable Distributed File System Using Object Disks. In *Proceedings of NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, 2003.

[12] Douglas J. Santry, Michael J. Feeley, Norman C. Hutchinson, and Alistair C. Veitch. Elephant: The File System that Never Forgets. In *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems (HOTOS)*, 1999.

[13] VMware. Understanding virtual machine snapshots in VMware ESXi and ESX. http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1015180.

[14] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 255–270, Boston, MA, December 2002.

[15] Wikibooks. QEMU/Images. http://en.wikibooks.org/wiki/QEMU/Images#Copy_on_write.