Towards a Representive Testbed: Harnessing Volunteers for Networks Research

Monzur Muhammad, Justin Cappos
Polytechnic Institute of New York University

1. ABSTRACT

A steady rise in home systems has been seen over the past few years. As more systems are designed and deployed, an appropriate testbed is required to test these systems. Several systems exist, such as PlanetLab, that currently provide a networking testbed allowing researchers and developers to test and measure various applications. However in the long run such testbeds will be unable to keep up and meet all the demands of many of the large scale modern day peer-to-peer systems. We outline the various challenges and essentials of a networking testbed and we provide an alternate networking testbed that is driven by resources that are voluntarily contributed. We talk about the various advantages and disadvantages of the Seattle system, an open source peer-to-peer computing testbed that has the potential to meet these demands. The testbed is composed of sandboxed resources that are donated by volunteers. Seattle has been deployed for about three years and supports many researchers who are interested in a networking testbed. The testbed consists of over 4100 nodes and is constantly growing. Seattle looks to grow and meet the demands of networking testbeds as they are made.

2. INTRODUCTION

Home computing has risen in popularity over the last few years as smartphones and tablets and various other devices have become an integral part of the Internet, driving an increase in demand for a suitable peer-to-peer testbed. Various different systems have emerged over time to meet these demands and produce a feasible testbed. Systems such as RON [3], PlanetLab [28], and VINI [8] are some of the more popular systems that have been widely used by researchers. These systems allow users to test network applications on them in order to verify and validate their functionalities.

However, all of these testbeds have similar designs and consist of a model of dedicated resources. Although this design may be suited for evaluating most applications, it may still restrict the user from performing various different tasks. The many limitations of existing testbeds have been well noted [36], and attempts have been made to improve and solve these challenges [29, 39, 14]. Despite the growing importance of home systems, networked testbeds have unfortunately not kept up in supporting the various peer-to-peer system designs. We provide three different limitations in the existing testbeds and how these limitations may need to be addressed in order to have a viable peer-to-peer testbed.

2.1 Limitations of Current Testbeds

2.1.1 Representation of real users

Many of the current testbeds are primarily deployed on universities and are akin to the same network settings, thus lacking in diversity. Similarly, other testbeds are deployed on dedicated servers that typically do not accurately model users and network trends of every day machines [14, 37, 42, 7]. Machines in the universities often consist of high end nodes with a very high speed Internet access. Even though high-speed Internet access has become more common over the past few years, many end-users still own average machines with standard Internet connection and only mediocre amounts of bandwidth. Furthermore residential nodes consists of wireless nodes, NAT devices, and many other network diversity that are not found in many testbeds. A diverse set of devices such as laptops and various other handheld devices have also become more popular over the last few years that have a rich mobility and availability pattern that cannot be represented by machines with static IP addresses [9, 6]. The dedicated machines in the various testbeds however fail to take into account the wide diversity that is seen in the real world. Simulating experiments on a system like PlanetLab will often not result in the most accurate representation of real world users [20]. Understandibly it is easier to administrate a controlled system. However to understand the behavior of home system, a diverse set of nodes and configurations is required. This is essential to understand peer-to-peer systems. In this paper we present a system that uses actual user machines as part of the testbed in attempts to get a better representation of the real world, thus respresnting the Internet as a testbed.

2.1.2 Limited scalability issues

Current peer-to-peer systems and cloud computing systems have somewhere between ten thousand to about a million nodes [2, 38, 15]. However, existing testbeds currently do not have the capacity to handle systems this large, as they mostly rely on dedicated servers and machines. In order for these testbeds to expand, new hardware and resources must be obtained, thus placing an upper bound on its capability to grow. Furthermore maintining individual nodes in the testbed becomes a challenge as the testbed grows. This means these testbeds cannot be reliably used to discover the complete behavior of many peer-to-peer systems. Thus we propose a testbed which consists of volunteered resources and has the potential to scale with demand. Popular peer-to-peer systems that need a testbed that meets their de-

mand can themselves donate resources in order help increase the size of the testbed as well as add resources that meet their criteria. The growth of the testbed will also attract other applications and many researchers who are interested in running experiments on a large scale testbed. Donated resources will be maintained by the donors, removing the necessity for a single administrator to manage the entire system. Since the size of the testbed will grow proportionally to how much resources are donated, it will encourage developers and researchers to donate enough resources in order to test their system at a large scale. This will encourage and attract not only big systems, but individuals and researchers as well, who would like to run experiments on a large scale, but are unable to do so due to insufficient resources.

2.1.3 *Deployment of testbed*

Internet testbeds were promoted in part because the research community values the importance of realistic deployment. However, today's testbeds serve few end-users and do not concern themselves with attracting them. When an application is lucky enough to attract users (e.g. Coral [18]), it begins to adversely impact the testbed by over-utilizing the available resources. This leaves researchers whose prototypes are deployed on PlanetLab with no means by which they may release their software into the wild. Some researchers do build and release their systems directly to end-users [27, 13]. However, this methodology has a high bar for success and limits researchers to developing systems with capabilities that appeal to end-users (e.g. file-sharing).

2.2 Are Volunteer Resources Plausible?

A question that may come to mind is whether resources donated by end users and researchers are capable of running the various tests and experiments. Certainly a testbed consisting of contributed resources has the potential to grow, however the resources that are donated will vary widely. In order to determine whether an end-user machine has sufficient resources to build a testbed on, we looked at some of the existing testbeds in order to understand how much resources are consumed by current experiments. We analyzed the PlanetLab system by using data from CoMon [24], a site that provides periodic statistics on resource usage of PlanetLab nodes. This information is publicly available and is updated at 15 minute intervals.

Resource usage data from the second week of February 2012 presented that on average a node with network activity has about 141 Kb/s upload rate and 120 Kb/s download rate. We also found that on average less then 100MB of physical memory is used by an application on 97% of the nodes (average memory used by a given slice, where a slice consists of several nodes).

As of January 2012, around 92% of the OS market share is owned by Microsoft Windows [22]. Of this share, roughly 47% consists of Windows XP and 36% of the market consists of Windows 7. Looking at the the OS share trend, there has been a steady rise of Windows 7 market share as Windows XP slowly decreases. Analysing the average resource consumptions of experiments run on PlanetLab and comparing it to the minimum hardware requirement of Windows XP[1] [41] and the requirements of Windows 7[2] [40] shows that end-user machines are capable of handling most experiments and tests that are run on popular networking testbeds.

[1] 233 MHz processor, 128MB RAM, 1.5GB free disk space
[2] 1 GHz processor, 1GB RAM, 16GB free disk space

Node Type	Quantity
Testbed	607
University	548
Home machines	1455
Unknown	1513
Total	4123

Figure 1: Seattle node types determined via reverse DNS lookups on systems that retrieve software updates.

2.3 Incentive for Volunteers to Contribute Resources

Unlike traditional testbeds, a volunteer-supported testbed must attract volunteers to have any resources. The volunteers will determine the heterogeneity of the testbed and two key non-technical challenges are how to attract volunteers, and how to retain them.

Volunteer based distributed computing projects have succeeded in attracting tremendous resources to their cause. For example, the DIMES project [33] has a user base numbering in the thousands, while SETI@home [5] is one of the longest running and best supported volunteer efforts with over 1.36 million computers in the system and a computational power rivaling that of the fastest supercomputer. The popularity of Wikipedia, Yahoo! Answers, and numerous other volunteer-supported online projects indicate that individuals are eager to join such projects and donate their resources in the form of time, knowledge, bandwidth, and other resources.

Similarly, we believe that similar levels of participation can be generated for a network testbed. We designed, implemented, and deployed a P2P network testbed called Seattle [31]. We attracted more than 4100 nodes to our system in just under two years of coming online. This total is broken down by categories in Figure 1. To attract volunteers we relied on word of mouth and incentivized the use of Seattle for teaching distributed systems and networking courses [11].

Once a volunteer decides to donate their resources, they must then be incentivized to remain, potentially increase their contributions over time, and hopefully induce others to volunteer. The technical challenge is in keeping experiments as unobtrusive as possible while the user is actively using the machine. The non-technical challenge in retaining users is to design an incentives strategy such that the incentive to contribute outweighs concern and the effort necessary to continue participation.

2.4 Attracting Users for Educational Purposes

Network testbeds are prone to attract researchers and software develoers, however it has the potential to attract a vast community of users in the education community as well. A wide peer-to-peer testbed that consists of nodes that are geographically diverse gives educators a way to teach students about the various challenges and difficulties of network programming. Students are able to produce software and test them on a set of diverse nodes allowing them to fully understand many networking concepts first hand. Seattle also provides a set of API calls [30] that simplify network

programming for beginners. It allows students to concentrate on the networking aspects of a program rather then worry about the complex set of networking API calls, thus making it easier for students to understand the issues of distributed and networked systems, instead of wasting time debugging complex network API calls. Educators are motivated to contribute to the testbed in order to add more diversity to it as well as enlarge it, allowing students to have a better learning experience and better understand network concepts. Furthermore, students that have a positive experience from running experiments on the testbed may also be motivated to continue working with it and donate personal resources to the testbed.

In the next section we outline the challenges facing a P2P network testbed design, and then present the Seattle architecture in Section 4. We detail one of Seattle's essential components – the programming language virtual machine – in Section 5. Section 6 describes how Seattle has been used, and details our experiences with the platform. We describe related work in Section 7 and conclude with Section 8.

3. CHALLENGES IN DESIGNING A PEER-TO-PEER TESTBED

In this section we provide an overview of the challenges a successful peer-to-peer testbed design must address.

Safety. The highest priority in designing and implementing an open testbed is end-user security. Volunteer machines must be able to execute general purpose code, and yet represent as little risk to the end-user as possible. Since a malicious researcher has the ability to run code on user machines, the testbed node hosting researcher code must provide a reasonable expectation of security *even with unpatched vulnerabilities on the underlying system.* Additionally, code executing on the testbed must not interfere with the performance of the volunteer's other applications or cause the volunteer to be notified about illegal or unwanted traffic [12].

Usability. The testbed must have tools to make it easy for researchers to use it. In particular, this involves tool support. For example, as PlanetLab grew, tools like PLuSH [1], Stork [10], and CoDeploy [23] have significantly simplified its use.

Scalability. The testbed must be able to grow to a large size. The management overhead must remain low regardless of the size of the testbed.

Simplicity. A peer-to-peer testbed software stack must operate without any volunteer intervention. Volunteers should not have to report problems or install security updates, and volunteering resources must be trivial.

Flexibility. Researchers must be able to write useful programs, and when permitted, the volunteer's applications must be able to interact with testbed code to allow production services to use the testbed. While an end host testbed may not be as general as PlanetLab, it should allow researchers to execute general purpose code.

Portability. A peer-to-peer testbed must run on a wide variety of operating systems and architectures. Nodes must be able to handle real-world network conditions like disconnected operation, IP address changes, and middle boxes like NATs and firewalls. Without portability, it is unlikely that the testbed will be representative of the variety of devices and networks making up the internet.

Incentives. There must be incentives for volunteers to join the testbed. Altruism can grow a testbed to a very large size [4, 5], but providing volunteers with other incentives, like desirable software features or better application performance, may attract even more participation. Similarly, there must be incentives to attract new researchers to the testbed.

In the next section we give an overview of the Seattle architecture and due to space constraints only briefly mention how Seattle addresses the challenges listed above. Section 5 gives a more in-depth look at the design of the Seattle virtual machine – a crucial component responsible for addressing the *safety* challenge.

4. SEATTLE ARCHITECTURE OVERVIEW

We term a program that a researcher wants to run on the testbed a **service**. A researcher's code is executed in a programming language **virtual machine** that runs python code, of which multiple separate instances can run on a system. Each Seattle node runs a **node manager** to mediate access to the system's set of VMs. This makes the system usable by allowing authorized researchers to perform actions such as starting, stopping, and reading the log from a virtual machine. Software on a Seattle node is updated using a push-based **software updater**, which updates the node manager, the virtual machine code, as well as the software updater itself. The software updater requires no user intervention and helps to address the simplicity challenge. A **service manager** provides researchers with a simpler interface to interact with the node manager. The service manager enables researchers to upload service code, to monitor the service, and to perform various other actions. A service manager addresses the usability challenge.

Seattle uses a **clearinghouse.** to mediate access to a shared pool of VMs. Researchers make requests to the clearinghouse for additional VMs on the testbed, which may be granted or denied according to a policy. Researchers may also choose not to use a clearinghouse, preferring to directly control the donated virtual machines.

Several DHT-like **lookup services** are leveraged to help a service manager or a clearinghouse to locate VMs under their control. A node manager associates its IP and port with the public key of all of the keys that control a VM. The service manager or clearinghouse can then look up its key to locate the virtual machines it controls. A scalable lookup service, like a DHT, is important for meeting the scalability challenge of the testbed.

Finally, there are other services that run inside and outside of the testbed that enhance its functionality. These services mediate connections to nodes behind NATs, ensure that network traffic is confined to the testbed, and help to reduce the testbed's load on public servers.

5. SEATTLE VIRTUAL MACHINE

The Seattle programming language VM can execute general purpose code with a few limitations intended to protect the host system from the experiment code. In this way, the VM most directly represents the tension between the challenges of flexibility, portability, and safety from Section 3. The Seattle VM is, in purpose, but not in implementation, analogous to a sliver on PlanetLab. For example, untrusted researchers are allowed to execute code on Seattle end-user systems because certain actions are disallowed through tech-

nical means instead of through policy. For example it is not possible to send ICMP packets, only TCP and UDP. A flaw in the VM implementation does not allow an attacker to escape the VM or consume undue resources on the end user's machine. The rest of this section overviews the design of the Seattle VM.

5.1 Programming Language.

The Seattle VM executes service code written in a subset of the Python language. In order to minimize the risk of bugs, the VM attempts to use only parts of the underlying trusted computing base that are stable, conceptually simple and widely used. For example, we allow use of a vanilla type of the Python interpreter's style of classes as well as only simple types; we do not allow classes that subclass basic types, provide their own namespace storage mechanisms, or utilize other rarely used or new functionality.

The VM language supports a subset of Python language primitives and constructs — it is, in fact, executed by the Python interpreter. The VM loads the service code as text and then uses a standard module built into the interpreter to build a parse tree. The VM verifies that the parse tree of the service only contains the supported subset of the language. If there is a disallowed language construct, the service code is rejected *without executing*.

The VM verifies multiple aspects of the Python interpreter before it begins executing service code. Built-in Python functions that are allowed are checked to ensure their signatures (i.e. arguments) are as expected. This is done to prevent changes to built-in functions due to different interpreter implementations or versions resulting in unintended functionality being exposed to untrusted researcher code.

The set of allowed built-ins consists of 87 items obtained directly from Python's interpreter. This includes definitions of constants such as True and exceptions (53), type conversion functions like float or chr (15), math functions like max and pow (8), as well as miscellaneous Python operations like len and range (11). Notice that operations like import (which includes code from another module), eval, and exec are not allowed because verifying their safety is known to be difficult. Further, to prevent escape of user code from within the VM we minimize the trusted computing base by employing a novel layering approach to move numerous standard libraries out of the trusted computing base. [12].

5.2 API

The set of built-in Python functions mapped into a VM excludes all functions that write data to disk, utilize network bandwidth, or perform other complex tasks. To handle these tasks, we provide a high-level API containing 17 API calls with another 14 calls accessible through objects. This provides programmers with the ability to read and write files on the disk, start threads, obtain locks, and send TCP and UDP traffic, and other abilities [30].

The API provides a high-level interface and has calls like `send_UDP_packet` and `getmyip` instead of calls like `bind` and `setsockopt`. This high-level interface makes it easier to infer the programmer's intent, which simplifies reasoning about security, and prevents exploitability of low-level bugs such as many syscall vulnerabilities. For example, our API does not allow the researcher to directly set socket options. It therefore protects against almost all exploits in `setsockopt`, which have been previously exploited in multiple operating systems (CVE-2004-0370, CVE-2004-0424). Some of our API calls will indirectly set socket options on some operating systems to provide consistent behavior across operating systems. However, our API implementation will only call a specific set of calls in a specific order. This is more secure, but less flexible than allowing the service to pass arbitrary socket options to the underlying OS.

5.3 Resource Accounting and Restriction

It is essential that services are performance isolated from the other applications on an end-user's machine. The resource accounting and restrictions layer of the Seattle VM tracks 17 resources, including CPU, memory, total disk space, read and write rates for the network and disk, TCP and UDP ports, and a few others [30]. If the service attempts to use a resource that is not assigned to the VM, such as a disallowed UDP port, the layer will prevent access to it. If the service attempts to consume too much of a resource, the call will either be denied or, for rate limited resources, the call will be delayed.

Most actions that consume resources are API calls that pass through the the resource accounting and restrictions layer. However, CPU and memory may be consumed without making any API calls. These resources are monitored by a separate thread or process (depending on the OS). If a service's memory use becomes too high, the service is terminated. To meter CPU use, the service is checked periodically. If the CPU use exceeds the allowance, the service's execution is temporarily suspended to compensate.

The resource metering that is done by Seattle is based upon hard resource limits. This means that a VM is expected to have a fixed amount of the resources on the underlying user's machine *regardless of the other programs or VMs that are executing*. This ensures that experiments only minimally conflict with user programs.

6. EXPERIENCES WITH SEATTLE

In this section, we describe the current state of the Seattle testbed, including its use by educators and researchers. Then we describe our ongoing efforts to grow the scale of the testbed by involving developers.

6.1 Current State of Seattle

The Seattle testbed is deployed on a wide variety of systems. As Figure 1 shows, from Jan 2010 to Jan 2011, 4123 unique IP addresses have retrieved update information from our software update server (omitting indexing bots). This does not mean that there are this many actual Seattle nodes, however, as there may be multiple nodes behind a single public IP. Also, nodes may change IP addresses or may have uninstalled the software. Additionally, a significant number of Seattle installations are behind NATs and firewalls. The NAT traversal service has been contacted by several hundred unique Seattle installations. Thus the Seattle testbed is comprised of a diverse set of resources that come from different sources.

We began with an in-house deployment on about 10 heterogeneous systems including Linux, Windows, Mac, and even mobile devices. These provided baseline tests for the portability of our virtual machine implementation and the remainder of the Seattle environment. Once we were comfortable with a small number of diverse nodes, we deployed onto all of PlanetLab. Educational use drove our software's

adoption at both universities and with home users. In the meantime, we have continued to aggressively pursue other deployment avenues. For instance, we have deployed Seattle onto buses through a partnership with Dome [35], although these nodes have not been made available on our public clearinghouse.

We currently run two clearinghouses, a public clearinghouse and a beta clearinghouse. The beta clearinghouse is used internally to test nodes and network environments that are not ready for production use, such as our containment service and buses. The beta clearinghouse previously contained nodes behind NATs or firewalls, but as we vetted our support for these systems, we moved them to the public clearinghouse and made them available to end-users.

The public clearinghouse is available for anyone to sign up for an account [32]. The clearinghouse uses a resource incentives model where each computer a researcher adds to the clearinghouse gives the researcher access to ten virtual machines at a time, generally on different physical machines. This is sustainable because each virtual machine consumes only about one eighth of the donated resources and all resources aren't fully utilized at all times. This provides incentives for educators and researchers to grow the testbed in exchange for a large and diverse set of resources

6.2 Research Use

Currently the Seattle testbed is being used by research groups in the US, Canada, and Europe. Despite the fact that development of Seattle started in July of 2008 and it was not made available to researchers until June of 2009, some interesting applications have already been constructed. In particular, researchers have found Seattle useful for accurate modeling of the diverse networking properties of Internet hosts. Here we give two example applications deployed on Seattle.

HuXiang: This service leverages the distributed nature of Seattle to implement an end-user-based web server. The server instances run on Seattle nodes and register a common domain name with a dynamic DNS service, or announce the same key in another lookup service. When their browser resolves the domain name or looks up the key, users are directed to one of the server instances, with the total load distributed over all participating nodes. Since nodes may join and leave the HuXiang formation over time, the web content can also evade IP address filters.

TryRepy: This web-based IDE allows users to write simple Repy programs and evaluate them without having to install Seattle. It allows users to try out the Repy programming language and get a feel for the language before writing complex applications. TryRepy itself runs on Seattle nodes and deploys the user written code onto various nodes in order to evaluate and return the result to the user. The code that the user inputs to evaluate is run in the safe sandboxed environment that Seattle provides.

6.3 Educational Use

We made a significant effort to make it as easy as possible to use the platform in an education context [11]. Seattle provides instructors with an environment where students can safely and easily experiment with real-world distributed systems. To assist instructors we created educational resources in the form of tutorials and references to teach how to use the platform and our Python subset. Seattle has been used in over 18 networking or distributed systems classes. The students have helped us tremendously to debug our documentation and implementation as they discovered corner cases. The experience we have gained from instructional use has been invaluable.

6.4 Large-Scale Adoption

To reach a larger scale, we aim to expand our outreach beyond educators and researchers. We feel that the existence of a safe, peer-to-peer application platform where resources can be shared, data and tasks distributed, and new application space explored will be of interest to software developers, including those developing for profit. In the case that a piece of software has a Seattle dependency, it will have Seattle bundled along. This will result in increased utility as the user base grows.

Whether software developers would choose to participate in an existing clearinghouse, such as the one we maintain, or would prefer to create a private clearinghouse is unclear. However, we do feel that there is a strong incentive for developers to have their software participate in an open clearinghouse rather than a private one as this will allow an application to benefit from greater network diversity and for applications with different resource bottlenecks to efficiently share resources.

In addition to providing users with software or other benefits, we support altruistic users. Relying on the altruism of Internet users has paid huge dividends for other computing efforts [4, 5]. In future work, we intend to adopt similar mechanisms to make our platform more attractive to this user base.

7. RELATED WORK

Whereas other systems have been developed to harness volunteers, none have aimed at creating a scalable general-purpose testbed. Volunteer computing efforts like BOINC [4] and SETI@Home [5] leverage unused compute resources on end-user computers. Access to volunteer computing resources is tightly controlled because developers are given low-level access to the machine without any security-conscious restrictions. Also, a volunteer computing platform will usually only execute when the system is idle so resource isolation is not a major concern.

There have also been a variety of efforts to gather measurements from end user machines [34, 25]. These systems do not allow researchers to measure paths between users (unless those intersect) or provide a mechanism for in-situ experimentation.

Grid computing [17] and testbeds like PlanetLab [26] leverage resources on dedicated machines. SatelliteLab [14] sends traffic from end-user systems through PlanetLab nodes to emulate home node link characteristics. This provides limited fidelity because traffic does not flow directly between end nodes. There have been efforts in FIRE [16] and GENI [19] to increase the scale and diversity of testbed resources. Seattle is involved in GENI, where it is called the Million Node GENI project [21], as one of the only projects that does not rely on dedicated hardware.

Like Java and Lua, Seattle's virtual machine is a programming language VM – it runs as a process and requires an interpreted language. It has many safety and security features that are similar to languages like Java. However,

compared to other programming language VMs the Seattle VM has a much smaller trusted computing base due to its small API and use of security layers [12]. The Seattle VM also provides stronger resource isolation guarantees.

8. CONCLUSION

In this paper we identified the various problems with the traditional design of building internet testbeds relying on dedicated resources. We motivated the need to consider new testbed designs and presented evidence for why a volunteer supported testbed of end hosts is a viable design choice. We then presented a set of challenges for such a design, and gave an overview of the design and our experiences with Seattle – a peer-to-peer testbed that we developed and deployed on over four thousand computers world-wide. We hope that this paper informs future progress in the research area of internet testbeds.

9. REFERENCES

[1] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, and A. Vahdat. Remote Control: Distributed Application Configuration, Management, and Visualization with Plush. In *Proc. 21th LISA (LISA '07)*, Dallas, TX, Nov 2007.

[2] Amazon EC2 - Amazon Web Services @ Amazon.com. `http://www.amazon.com/gp/browse.html?node=201590011`.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th SOSP*, pages 131–145, Banff, Alberta, Canada, Oct 2001.

[4] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Commun. ACM*, 45(11):56–61, 2002.

[6] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing user behavior and network performance in a public wireless lan. In *SIGMETRICS '02*, pages 195–205, New York, NY, USA, 2002. ACM.

[7] S. Banerjee, T. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes.

[8] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. *SIGCOMM CCR*, 36(4):3–14, 2006.

[9] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *IPTPS'03*, page 256. Springer-Verlag New York Inc, 2003.

[10] J. Cappos, S. Baker, J. Plichta, D. Nyugen, J. Hardies, M. Borgard, J. Johnston, and J. Hartman. Stork: Package Management for Distributed VM Environments. In *Proc. 21th LISA (LISA '07)*, Dallas, TX, 2007.

[11] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. *SIGCSE Bull.*, 41(1):111–115, 2009.

[12] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson. Retaining sandbox containment despite bugs in privileged memory-safe code. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 212–223, New York, NY, USA, 2010. ACM.

[13] D. R. Choffnes, F. E. Bustamante, and Z. Ge. Crowdsourcing service-level network event detection. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 387–398, New York, NY, USA, 2010. ACM Press.

[14] M. Dischinger, A. Haeberlen, I. Beschastnikh, K. P. Gummadi, and S. Saroiu. Satellitelab: adding heterogeneity to planetary-scale network testbeds. *SIGCOMM CCR*, 38(4):315–326, 2008.

[15] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a million user dht. In *IMC '07*, pages 129–134, New York, NY, USA, 2007. ACM.

[16] FIRE: Future Internet Research and Experimentation. `http://cordis.europa.eu/fp7/ict/fire/`.

[17] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[18] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *NSDI'04*, March 2004.

[19] GENI: Global Environment for Network Innovations. `http://www.geni.net/`.

[20] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In *Proc. 4th NSDI*, Cambridge, MA, Apr 2007.

[21] MillionNodeGENI – GENI: geni – Trac. `http://groups.geni.net/geni/wiki/MillionNodeGENI`.

[22] Netmarketshare. `http://www.netmarketshare.com/os-market-share.aspx?qprid=9`, Accessed February 14, 2012.

[23] K. Park and V. S. Pai. Deploying Large File Transfer on an HTTP Content Distribution Network. In *WORLDS'04*, San Francisco, CA, Dec 2004.

[24] K. Park and V. S. Pai. Comon: a mostly-scalable monitoring system for planetlab. *SIGOPS OSR*, 40(1):65–74, 2006.

[25] V. Paxson, A. Adams, and M. Mathis. Experiences with NIMI. In *SAINT'02*. IEEE Computer Society Washington, DC, USA, 2002.

[26] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. HotNets–I*, Princeton, NJ, Oct 2002.

[27] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *Proc. 4th NSDI*, Cambridge, MA, Apr 2007.

[28] PlanetLab. `http://www.planet-lab.org`. Accessed February 15, 2012.

[29] H. Pucha, Y. C. Hu, and Z. M. Mao. On the impact of research network based testbeds on wide-area experiments. In *IMC '06*, pages 133–146, New York, NY, USA, 2006. ACM.

[30] RepyLibrary – Seattle – Trac. `https://seattle.cs.washington.edu/wiki/RepyApi`. Accessed February 15, 2012.

[31] Seattle: Open Peer-to-Peer Computing. `http://seattle.cs.washington.edu/`. Accessed February 15, 2012.

[32] SeattleGENI - Global Environment for Network Innovations using Seattle. `https://seattlegeni.cs.washington.edu/`. Accessed February 15, 2012.

[33] Y. Shavitt and E. Shir. Dimes: let the internet measure itself. *SIGCOMM CCR*, 35(5):71–74, 2005.

[34] C. Simpson, D. Reddy, and G. Riley. Empirical models of TCP and UDP end-user network traffic from NETI@ home data analysis. In *PADS'06*, pages 166–174, 2006.

[35] H. Soroush, N. Banerjee, A. Balasubramanian, M. D. Corner, B. N. Levine, and B. Lynn. DOME: A Diverse Outdoor Mobile Testbed. In *HotPlanet*, June 2009.

[36] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using

planetlab for network research: myths, realities, and best practices. *SIGOPS OSR*, 40(1):17–24, 2006.

[37] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for network research: myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 40(1):17–24, 2006.

[38] Storm worm: More powerful than Blue Gene? - ZDNet UK. `http://news.zdnet.co.uk/security/0,1000000189,39289226,00.htm`.

[39] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02*, pages 271–284, New York, NY, USA, 2002. ACM.

[40] Windows 7 system requirements. `http://windows.microsoft.com/en-US/windows7/products/system-requirements`, Accessed February 15, 2012.

[41] How to upgrade to windows xp. `http://support.microsoft.com/kb/316639`, Accessed February 15, 2012.

[42] H. Zheng, E. K. Lua, M. Pias, and T. G. Griffin. Internet routing policies and round-trip-times. In *In PAM*, 2005.