# The GENI Experiment Engine

Andy Bavier
PlanetWorks, LLC and
Princeton University
acb@cs.princeton.edu

Jim Chen
iCAIR,
Northwestern University
jim-chen@northwestern.edu

Joe Mambretti
iCAIR,
Northwestern University
j-mambretti@northwestern.edu

Rick McGeer
Communication Design Group
SAP Labs, USA
rick.mcgeer@cdglabs.org

Sean McGeer
Duck Syrup
seanmcgeer@gmail.com

Jude Nelson
Princeton University
jcnelson@cs.princeton.edu

Patrick O'Connell
Duck Syrup
patrickboconnell@gmail.com

Glenn Ricart
US Ignite
glenn.ricart@us-ignite.org

Stephen Tredger
University of Victoria
stredger@gmail.com

Yvonne Coady
University of Victoria
ycoady@cs.uvic.ca

## ABSTRACT

We describe the GENI Experiment Engine, a Distributed-Platform-as-a-Service facility designed to be implemented on a distributed testbed or infrastructure. The GEE is intended to provide rapid and convenient access to a distributed infrastructure for simple, easy-to-configure experiments and applications. Specifically, the design goal of the GEE is to permit experimenters and application writers to: (a) allocate a GEE slicelet; (b) deploy a simple experiment or application; (c) run the experiment; (d) collect the results; and (e) tear down the experiment, starting from scratch, within five minutes. The GEE consists of four cooperating services over the GENI infrastructure, which together with pre-allocated slicelets and a pre-allocated network offers a complete, ready to use, sliceable platform over the GENI Infrastructure.

## 1. INTRODUCTION AND MOTIVATION

The Global Environment for Network Innovations (GENI)[3] is a distributed Infrastructure-as-a-Service (IaaS) platform with deeply programmable networking across a nationwide layer-2 software-defined network. With small clusters at over 50 sites, it is an ideal platform to construct, deploy and run arbitrary distributed systems and networking experiments.

The extreme configurability and flexibility of GENI, which is its great strength, also makes it complex. GENI experimenters and users must specify details of each node, including OS and programming environment, and details of the network toplogy between nodes. However, many experiments are agnostic on these various points: for example, any experiment that could have run happily as a Planetlab[12] slice has its OS and network predetermined (Fedora Core, and whatever the public Internet says). For these experiments and applications, GENI's rich feature set and extreme configurability is a barrier to entry.

The GENI Experiment Engine (GEE) is designed as a restricted, easy-to-use programming platform on GENI. Our fundamental mantra is that it should be easier to configure, deploy, and run an experiment than it is to design and write it. In the extreme, this translates into the "five-minute rule": one should be able to compile, deploy, and run a "Hello, World" experiment in five minutes.

The GEE is intended for several purposes:

1. To permit high-in-the-stack distributed systems experimenters to use GENI without having to allocate virtual machines, configure virtual networks, write Resource Specifications (RSPECs), configure VMs, etc.

2. To provide single-pane-of-glass control of an experiment from the user's desktop.

3. To provide a GENI-wide filesystem-like storage infrastructure for GEE Experimenters.

4. To provide a messaging infrastructure for GEE experiments.

5. To provide shared, application-level HTTP server access to the public Internet.

These features were derived from an analysis of a number of applications and demonstrations of GENI that we had built over the years, notably TransCloud[2] at GEC-10 and TransGeo at GEC-16. After both of these demonstrations, the most common question that we were asked was "How did you build all that stuff?" and "Can my experiment use the infrastructure you built for that system?" The latter question meant more than the various GENI aggregates we used, and in one case built; it was the specific set of application-level services that we built to undergird our demonstrations, and the deployment engines that we used to deploy our application across GENI.


978-0-9836283-9-2 

From these questions, the idea for the GEE was born. Specifically, we asked ourselves which chunks of our demonstrations could be re-used by other experimenters, and what tools and services would we have found useful in building and deploying these demonstrations; then we asked what barriers to deployment we encountered and how we could remove those.

The remainder of this paper is organized as follows. In Section 2, we describe the process of using GEE: how a user will allocate, use, and tear down a GEE "slicelet". In Section 3, we describe the architecture and component services of the GEE. In Section 4, we describe related work in both the testbed and cloud arenas. In Section 5, we describe the current status of the GEE and its future.

## 2. RUNNING A GEE EXPERIMENT

The easiest way to get a feel for an architecture like GEE is to consider its usage. To use the GEE, a user logs in to the GEE portal using her GENI credentials. The GEE portal stores no user information or credentials; instead, OpenID[13] is used to call back to the GENI portal, and the user's returned email is used as the userid for the purposes of the GEE Portal. The user is then directed to a dashboard, where, with the click of a single button, she can allocate a GEE Slicelet. When this process is completed (typically instantaneously), a download link to a small zip file appears on her dashboard. The user can then download the file to his computer. The zip file contains four files:

1. The slicelet's private key
2. A Python file, in a standard format, containing slice-specific constants for use in the experiment. In particular, it has at the IP addresses for the private network associated with the slicelet, under names of the slicelet sites.
3. A Fabric[7] file, containing the slice name and private key, that the user can use to deploy software or configure the slivers on the slice.
4. A README file.

Of the four items, only the first is required to access the slice. The remaining three are convenience items to get "Hello, World" up and running.

Once the user has the private key and has stored it in his .ssh directory, she is immediately able to ssh into slivers in the usual fashion, and configure them in the usual way. A user will also be able to use any ssh tool of her choosing to populate or control her slice. However, use of the enclosed Fabric file makes upload and execution as easy and quick (roughly, as easy as uploading a Python program to the Google App Engine).

Fabric is our solution to single pane-of-glass control of a slicelet. It is simply a Python wrapper around ssh commands, which automates the execution of both remote and local commands. We have pre-loaded the Fabric file with a number of commands to both introduce the user to Fabric and to give them out-of-the-box functionality on the site. Typing: "fab nmap" runs a script on each host that reports the reachable IP addresses on the private network.

Once the user has completed her experiment, she tears it down by using the "free slicelet" button on the GEE portal.

It's important to note that no configuration of the slivers in the slicelet is required: the user simply runs her experiment. Indeed, if the software the experiment requires is pre-installed on a typical Fedora Core distribution, the user need not necessarily upload any software at all.

## 3. ARCHITECTURE AND IMPLEMENTATION OF THE GEE

The GEE is configured as a set of four services: a Compute Service, which allocates GEE "slicelets" and configures them; a Storage Service, which offers a filesystem interface onto a distributed store; a Message Service, which offers a simple mechanism for slivers within a slicelet to send messages to each other; and a Reverse Proxy, which offers outbound HTTP access to slivers within a slicelet. These services all rely on a persistent, GENI-wide layer-2 network, the GEE Network.

Of the four services, the GEE Compute Service is offered through the GEE Portal and is an overlay on InstaGENI[1] PlanetLab: it hands out "slicelets" of standard PlanetLab virtual machines, with some additional software pre-installed. The GEE Storage Service is offered in slivers itself – a Python library is pre-loaded into the user's slicelet, which presents a filesystem API to the end user. The library itself then makes REpresentational State Transfer (REST) calls to a network of storage proxies in the GEE Storage Service to store and retrieve data. The Message Service is simply a server which can be loaded into the slicelet, and a client library; a user activates the server on whichever nodes in the slicelet she prefers through a simple fabric command. The Reverse Proxy Service runs in a slicelet, and controls HTTP ports on the routable interfaces of the GEE nodes. A slicelet registers to use the reverse proxy service through the GEE portal. After that, HTTP requests to that slicelet's sliver are routed by the reverse proxy to the sliver's http server.

### 3.1 The GEE Compute Service

The GEE Compute Service is a simple overlay on the InstaGENI PlanetLab infrastructure. InstaGENI PlanetLab was always envisioned as a subservice running over ProtoGENI[15] on the InstaGENI racks, and a convenient way for GENI users to run VM-based ex-


978-0-9836283-9-2 

periments. We have augmented this for GEE in three ways:

1. Enhancements to the standard PlanetLab VM to offer GEE and other services pre-installed;
2. Pre-allocation of GEE slicelets
3. Use-once credentials for access to GEE slicelets

The "five-minute rule" has dominated our design consideration. Delay in use of PlanetLab slices after allocation is due to sliver configuration and key propagation. To minimize these times, we pre-allocate and populate the slice. A bank of unused GEE slices is maintained at all times, with the GEE Programming Environment Installed. At a minimum, we pre-install: Python; pip; the GEE Filesystem Python library; yum; and the use-once public key. A number of other services, such as the GEE Message Service, can be activated with a simple Fabric command. The effect of all of this is that the slice is usable as soon as the use-once slice private key is downloaded; the user won't have to wait for slice configuration or key propagation.

Note that even though we're providing the use-once slice private key as a convenience, for bootstrap and for immediate usability, the user will be able to install keys of her choosing on the slice once she has access to it. One can even remove the use-once key if desired, though this is not recommended. Some actions void warranties.

We used a use-once, or "burner" key for two primary reasons: speed and security. Speed is obvious: we have pre-propagated the key. Security is nearly as obvious: if a user's slicelet is compromised, or the use-once key is discovered, all that is compromised is the user's slicelet. The GEE portal retains no credential from the user of any sort, and therefore cannot be a vector for compromise of any user information or credentials. Similarly, compromise of a user's ssh key won't result in an attacker gaining access to a GEE slicelet.

Use-once keys are the infrastructure equivalent of hotel room cardkeys; they are allocated when the slicelet is instantiated, used only to access the slicelet, and are destroyed when the slicelet is de-allocated. As a result, they come with many fewer security concerns than do standard keys, just as a hotel is completely unconcerned with travelers departing with room cardkeys in their pockets.

## 3.2 The GEE Storage Service

### 3.2.1 Overview

A file system consists of a block storage layer, and a metadata service which groups blocks into files, implements naming and directory structures and enforces access control. In this project, we will use an existing storage infrastructure with a Representational State Transfer (REST) interface to implement the primitive block-storage, and a global, universally-accessible database system with a REST interface to implement the metadata service. We will use the Swift object store as the storage layer, and the Syndicate file system to implement the file system metadata service. Using Swift and Syndicate we can create a distributed, highly available, accessible file system.

### 3.2.2 Requirements

The GENI Experiment Engine File System (GEE FS) is designed to be an easy-to-use file system provided on all GEE slices. We need the file system to be accessible both inside and outside experiments to allow users to access stored data from inside and outside GENI experiments. The GEE FS provides an accessible, persistent, environment for all GENI experiments. Since we want to make the GEE FS as easy-to-use as possible we have the following design goals:

1. Unix-like semantics
2. Convenient, reliable, distributed storage
3. Accessible from any GENI experiment
4. Runs on any reasonable host backend
5. Exposed API
6. Web interface for file browsing

The GEE FS is built using OpenStack Swift[16] (an open source Amazon S3-like service) nodes as a backend for Syndicate[11], a wide-area file system being developed at Princeton University and ONLab. Syndicate handles metadata in the file system as well as access control, versioning, and replication, while providing a familiar Unix like interface. Syndicate allows us to use multiple backend services distributed around the GENI network. The remainder of the section looks at each component of the file system in more detail.

### 3.2.3 Metadata Server

The most integral component of the file system is the Metadata Server (MS), which handles all file system metadata requests. For this we need a reliable service that can handle a lot of concurrent connections, and is easily accessible. We use the Syndicate MS[11], which is implemented as a Google App Engine application and stores its data in BigTable. By using Google App Engine, we get efficient app scaling under various loads, as well as efficient key-value lookups in BigTable. Users and Groups are handled by the MS restricting what a given user can access locally through the file system client. Users register an account through the file system client and provide a password for authentication. The password is used to authenticate subsequent file system requests.


978-0-9836283-9-2 

#### 3.2.4 Storage Service

Apart from the MS, Syndicate has client processes and storage processes. The storage service is a Python process that runs on remote nodes and act as a translator between Syndicate, and the storage service being used. Syndicate writes blocks of data to a storage service, and replicates the data for data durability. In our case we use Swift installations running in GEE slivers as our storage service. Swift is accessed via HTTP and also provides a Python API which allows easy integration with Syndicate. Storage services can be added and removed from Syndicate on the fly as the MS handles the actual layout of data (and its replicas) which gives us the flexibility to grow our storage capacity to meet the demands of GENI users. Additionally other storage providers (S3, dropbox, even the local file system) can be integrated into the file system.

#### 3.2.5 Client

The file system client exists as both a Filesystem in Userspace (FUSE)[8] and a Python module. The FUSE module allows users to mount the file system directly on any Unix-like system. The Python module allows Python processes to bypass FUSE and access the file system directly. The API allows clients to control the physical location of their files.

#### 3.2.6 File Browser

The file browser is implemented as a web interface using the Lively Web[9]. It allows users to upload, remove, and reorganize files in the file system. Lively gives the user the ability to customize the GUI via drag and drop on a canvas to create a highly customizable interface. The file browser will expose the physical location of files so users can move files both logically within the file system hierarchy, and physically onto different storage servers.

### 3.3 The GEE Message Service

The GEE Message Service is used to route job control messages within a slicelet; this is a common feature of many Cloud systems, and as a result a number of systems are available. We searched for one that is extremely simple, configures automatically, has a rich set of client libraries, can be enabled on the server side with a simple `service start` command, and whose use is well-documented.

We chose Beanstalk. Beanstalk has libraries in a wide variety of client languages, notably including Python. It installs as a service on Fedora, with a configurable port. It has an extremely simple put/get interface and supports a wide variety of use models, including pub/sub.

As with many Message Service systems, Beanstalk is configured for a single-tenant environment. Its use mode is not that a multi-tenant provider offers messaging-as-a-service, but rather that each job or service instantiate its own messaging server accessable only from its own nodes: security is assumed at the network, not the service, level. This dictated our deployment choice: rather than instantiating a GEE- or GENI-wide messaging service, we chose to offer the experimenter a Fabric command to turn the service on in the appropriate slivers, and choose the appropriate server site.

### 3.4 The GEE Reverse Proxy Service

Intra-slicelet traffic on the The GENI Experiment Engine will primarily be through a network private to the slice, which is isolated from the public Internet. Routable IPs are notoriously scarce on PlanetLab nodes, and GENI member institutions have been unwilling or unable to devote large banks of routable IP addresses to GENI slices. A GENI rack is capable of running well over 100 slivers, and we believe that 256 is achievable on the InstaGENI racks. We would need a /24 per site to accommodate these slivers, and at many sites we're lucky to get a /27. Clearly, we cannot count on being able to give a routable IP to each sliver.

Though GENI's private network suffices for intra-slicelet traffic, a number of PlanetLab slices and services offered distributed public services. The most notable of these were the Content Distribution Networks (CoDeeN and Coral)[17], End-System Multicast[6], and the Distributed Hash Tables[14]. Clearly, for such services to use the GEE, some method must be found to enable public-facing services at each site.

We don't have enough IP addresses to offer each public-facing service its own routable IP, and it isn't really feasible to assign each its own port: an http service that isn't on port 80 faces multiple logistical problems, from firewalls to configuration of client-side software. The solution we hit upon was to multiplex the http ports and isolate at the URL level, enabled enforced by the GEE Reverse Proxy.

The GEE Reverse Proxy Service operates a reverse proxy in a sliver on each GEE site. HTTP put, get, and post requests of the form `http://<hostname>/<sliceletname>/<request>` are caught by the reverse proxy and sent to the http server in the slicelet's sliver over the GEE private network; the returned value is sent back to the requester.

By default, the GEE Reverse Proxy Service is disabled for a slicelet, to prevent the slicelet's server from dealing with unanticipated requests. The experimenter selects proxy service for her slicelet from the GEE Portal dashboard; the portal then sends an authenticated request to enable proxy service for this slicelet to the reverse proxy. This is disabled on experimenter request or when the slicelet is destroyed.

### 3.5 The GEE Network


978-0-9836283-9-2 

The GEE Network is a private layer-2 network spanning the infrastructure on which the GENI Experiment Engine is deployed. Each GEE Sliver has a single interface on this network, with a `10.` address.

Since the network is pre-allocated, the user won't know the IP addresses of her slicelet until she acquires it, which means she won't be able to put the addresses in her code. This involves either hand-editing when she allocates the slicelet, or using symbolic addresses which are bound to values when the slicelet is allocated.

We define a Python library module in each slicelet, network.py. The programmer uses it simply by importing it into code. network.py defines constants, one per sliver, by symbolic location: so, e.g., `Northwestern = '10.64.136.1'`,

GEE Slicelet networks are not completely isolated. One of the major use cases for slices in PlanetLab was slices providing services for other slices: e.g., PsEPR[4] provided monitoring information and Stork[5] loaded images for other slices efficiently. The PlanetLab mantra for services was "put it in a slice", which led to a micro-kernel architecture for a distributed system: if it didn't absolutely need to be in the PlanetLab controller, it was in a services slice. This greatly simplified the design of PlanetLab, permitted experimentation in utilities and services, and contributed to the lifespan and maintainability of the PlanetLab infrastructure.

For these reasons, the GENI Experiment Engine is adopting the same design philosophy. The GENI Storage Service is deployed in a slicelet, as is the GENI Reverse Proxy Service. Our original intent was to offer the messaging service in a slicelet, but the requirement for a secure multi-tenant service restricted our choices and added unnecessary complexity to what was otherwise a simple, foolproof mechanism: hence our choice to add a service to the slicelet rather than offer a multi-tenant service in its own slicelet.

One difference between PlanetLab and GENI is the fact that GENI has a private network that today is used for intra-slice communication. Use of the private network is attractive for two reasons: conservation of port space on routable IPs and security. Public-facing services are under continual attack from botnets, something privately-deployed services need not protect against. Effectively, the GEE private network is a virtual intranet for GEE slicelets, and we intend to use it for GEE-specific services.

Implementation of the GEE Network evolves as GENI itself evolves. Our demonstration network used a temporary circuit from Ion, which is unsuitable for production use. Our first production network uses a GENI-wide VLAN. In June 2014, we instantiated a version of the GEE integrated with the Virtual Topology Service under development by Nick Bastin. GEE-on-VTS is the long-term architecture for the GEE network Eventually, we may be able to expose SDN functionality to GEE slicelets, as the northbound API for network operating systems becomes firm.

## 4. RELATED WORK

The GENI Experiment Engine is a Platform-as-a-Service (PaaS) operated on top of an Infrastructure-as-a-Service (IaaS) base. In this, it is not unique. The Google App Engine is a very heavily-used PaaS offering on Google's infrastructure. Further, there are deployable PaaS offerings. OpenShift from RedHat is an offering which orchestrates application deployment on the public cloud and offers PaaS on the enterprise cloud. There are many other examples: after all, to a first approximation offering PaaS on an IaaS offering is simply populating component VM's with the appropriate programming environments and platforms and providing orchestration services, notably automated scalability.

Commercial PaaS offerings focus on scalability and automatically scaling applications. In the GENI context, this is not a consideration: we do not have arbitrary resources on any single rack to scale the application; for GENI applications, location matters far more than scalability. Our primary concern is communication across the wide area and network design, concerns that are not relevant for data-center orienter PaaS systems.

AptLab (https://www.aptlab.net/) is similar in spirit to the GEE: it is a set of pre-configured "profiles" (essentially, pre-defined slices) on Emulab, and is designed to get users up and running fast on Emulab.

## 5. STATUS AND FUTURE WORK

The GEE is being brought up and deployed in stages, as the various services mature. The GEE Portal is up and running on InstaGENI PlanetLab Central, and the GEE Compute Service is functional. We demonstrated the GEE Compute Service and the Fabric-based single-pane-of-glass experiment control at GEC 19[10]. The GEE Compute Service can be moved into full production as soon as GENI nodes can be obtained on a long-term basis and the GEE Network stabilized; both of these require negotiations with the owners of these resources, and we are working closely with the GENI Project Office to secure these resources.

The GEE Storage Service is nearly as mature. The integration between the Swift proxies and the Syndicate metadata service is complete, and has been tested on GENI and Emulab. We will be ready to do a beta deployment as soon as the resources (primarily, VM's on GENIRacks) are obtained on a long-term basis. The GEE Filesystem browser is under development and is expected to be completed by September 2014. The GEE Proxy Service and the GEE Message Service have both been tested on VICCI slices, and we will test in GEE Slicelets within the next month.

The functionality and stability of the GEE is because it is built on well-tested and deployed infrastructure services and off-the-shelf components. The base for our compute service is PlanetLab, a 24/7 infrastructure that has run continuously for more than a decade; for storage, we used Swift, the block store for OpenStack; for messaging , we used Beanstalk; and for single-pane-of-glass control, we used Fabric.

GEE lives light on the land. Our interface to PlanetLab is a few Planetlab shell scripts; to ID providers an OpenID callback. Our remaining services can be instantiated inside VM's. We should be able to bring up the GEE on any distributed infrastructure with key-based access to the allocated VM's.

## Acknowledgements

The authors thank Leigh Stoller and Rob Ricci of the University of Utah and Niky Riga and Mark Berman of the GPO for much logistical assistance in setting up and maintaining the GEE Slicelets; Marshall Brinn of the GPO and Nick Bastin of Barnstomer Softworks for productive conversations; Chip Elliott of the GPO for years of guidance; Joe Kochan and the staff at US Ignite; Patrick Scaglia, Sanjay Rajagopalan, and Carlie Pham of SAP/CDG for financial and logistical support and mentoring; and Jack Brassil of the InstaGENI project and HP Labs for invaluable help. This project was partially funded by the GENI Project Office under subaward from the National Science Foundation.

## 6. REFERENCES

[1] N. Bastin, A. Bavier, J. Blaine, J. Chen, N. Krishnan, J. Mambretti, R. McGeer, R. Ricci, and N. Watts. The instageni initiative: An architecture for distributed systems and advanced programmable networks. *Computer Networks*, 61(0):24 – 38, 2014. Special issue on Future Internet Testbeds - Part I.

[2] A. Bavier, Y. Coady, T. Mack, C. Matthews, J. Mambretti, R. McGeer, P. Mueller, A. Snoeren, and M. Yuen. Genicloud and transcloud. In *Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, FederatedClouds '12, pages 13–18, New York, NY, USA, 2012. ACM.

[3] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds - Part I.

[4] P. Brett, R. Knauerhase, M. Bowman, R. Adams, A. Nataraj, J. Sedayao, and M. Spindel. A shared global event propagation system to enable next generation distributed services. In *WORLDS '04*, 2004.

[5] 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 *The 21st Large Installation System Administration Conference '07*, 2007.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP '03*, 2003.

[7] Fabric api documentation. `http://docs.fabfile.org/en/1.8/`.

[8] Fuse. `http://fuse.sourceforge.net/`.

[9] Lively. `http://www.lively-web.org/`.

[10] R. McGeer. Gec 19 gee demo video. `https://www.youtube.com/watch?v=RDnWIqtatkA`.

[11] J. Nelson and L. Peterson. Syndicate: Democratizing cloud storage and caching through service composition. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 46:1–46:2, New York, NY, USA, 2013. ACM.

[12] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building planetlab. In *In Proceedings of the 7th USENIX Symp. on Operating Systems Design and Implementation (OSDI*, 2006.

[13] D. Recordon and D. Reed. Openid 2.0: A platform for user-centric identity management. In *Proceedings of the Second ACM Workshop on Digital Identity Management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM.

[14] S. C. Rhea. *Opendht: A Public Dht Service*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2005. AAI3211499.

[15] R. Ricci, J. Duerig, L. Stoller, G. Wong, S. Chikkulapelly, and W. Seok. Designing a federated testbed as a distributed system. In *Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, June 2012.

[16] Swift. `https://wiki.openstack.org/wiki/`.

[17] L. Wang, K. S. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the codeen content distribution network. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.