

NCL Testbed Documentation

25 June 2018

Contents

| | |
|---|----|
| What is NCL Testbed? | 7 |
| Important Links | 7 |
| NCL Testbed Glossary | 7 |
| Registering to use NCL Testbed..... | 9 |
| Who may apply for a NCL project or request a user account? | 9 |
| To Register | 9 |
| Requesting a New Project | 9 |
| Getting a User Account to Join an Existing Project | 10 |
| Note to Students..... | 10 |
| If you need assistance..... | 10 |
| Getting Help with NCL | 10 |
| What to do first if you are having trouble? | 10 |
| Frequently Asked questions | 11 |
| Why can't I log in? | 11 |
| Password Guidelines | 13 |
| NCL Testbed Usage Policy..... | 14 |
| Be a good citizen | 14 |
| Core Quickstart..... | 16 |
| What is NCL Testbed? | 16 |
| How does it work? | 16 |
| How do I get a NCL Testbed account? | 17 |
| How do I use NCL Testbed? | 17 |
| 1. Design the topology | 17 |
| 2. Create and start (allocate resources for) an experiment | 18 |
| 3. Generate traffic for your nodes | 18 |
| 4. View results by accessing nodes, modify the experiment as needed. | 18 |
| 5. Save your work and stop your experiment (release the resources) | 18 |
| Core Guide | 19 |
| NCL Testbed Environment | 19 |

| | |
|--|----|
| Basic Tutorial | 19 |
| Getting Started | 19 |
| Step 1: Design the topology | 19 |
| Step 2: Create & start a new experiment | 24 |
| Step 3: Access nodes in your lab environment | 28 |
| Step 4: View results and modify the experiment | 29 |
| Step 5: Configure and run your experiment. | 29 |
| Step 6: Save your work and swap out (release resources) | 29 |
| Why can't I log in to NCL Testbed? | 31 |
| Installing RPMs automatically | 31 |
| Installing TAR files automatically | 32 |
| Starting your application automatically | 32 |
| Notifying the start program when all other nodes have started..... | 33 |
| Setting up IP routing between nodes | 34 |
| Sample Topologies | 38 |
| Toy topologies..... | 38 |
| LAN..... | 38 |
| Ring | 39 |
| Dumbbell..... | 40 |
| Using Your Nodes | 41 |
| Know your NCL servers | 41 |
| Hostnames for your nodes..... | 41 |
| Logging into your Node | 42 |
| How do I install software onto my node? | 43 |
| How do I copy files onto my node? | 43 |
| I need root access!..... | 43 |
| My node is wedged! | 43 |
| I want to load a fresh operating system on my node | 44 |
| I need more disk space on my node | 45 |
| Per-Link Traffic Shaping ("linkdelays")..... | 48 |
| User Do's and Don'ts | 50 |
| Preserving our Control Network..... | 50 |
| Preserving our File System..... | 50 |

| | |
|--|----|
| Preserving CPU Cycles on users | 50 |
| NCL Testbed Commands | 51 |
| Transport Layers | 51 |
| Operational Commands | 51 |
| startexp: Start an NCL Testbed experiment | 51 |
| batchexp: Synonym for startexp | 52 |
| endexp: Terminate an experiment | 52 |
| delay_config: Change the link shaping characteristics for a link or LAN | 52 |
| modexp: Modify experiment | 53 |
| swapexp: Swap experiment in or out | 54 |
| create_image: Create a disk image from a node | 54 |
| eventsys_control: Start/Stop/Restart the event system | 55 |
| loghole: Downloads and manages an experiment's log files | 55 |
| os_load: Reload disks on selected nodes or all nodes in an experiment | 57 |
| portstats: Get portstats from the switches | 57 |
| node_reboot: Reboot selected nodes or all nodes in an experiment | 58 |
| expwait: Wait for experiment to reach a state | 59 |
| node_list: Print physical mapping of nodes in an experiment | 59 |
| expinfo: Get information about an experiment | 60 |
| node_avail: Print free node counts | 60 |
| node_avail_list: Print physical node_ids of available nodes | 61 |
| nscheck: Check and NS file for parser errors | 61 |
| NS Commands | 63 |
| TCL, NS, and node names | 63 |
| Captured NS file parameters | 64 |
| Ordering Issues | 65 |
| Hardware Commands | 66 |
| tb-set-hardware | 66 |
| IP Address Commands | 66 |
| tb-set-ip | 66 |
| tb-set-ip-link | 67 |

| | |
|---|----|
| tb-set-ip-lan | 67 |
| tb-set-ip-interface | 68 |
| tb-set-netmask | 68 |
| OS Commands..... | 69 |
| tb-set-node-os | 69 |
| tb-set-node-rpms | 69 |
| tb-set-node-startcmd | 70 |
| tb-set-node-cmdline | 70 |
| tb-set-node-tarfiles | 70 |
| Link Loss Commands..... | 72 |
| tb-set-link-loss | 73 |
| tb-set-lan-loss | 73 |
| tb-set-node-lan-delay | 74 |
| tb-set-node-lan-bandwidth | 74 |
| tb-set-node-lan-loss | 75 |
| tb-set-node-lan-params | 76 |
| tb-set-link-simplex-params | 76 |
| tb-set-lan-simplex-params | 77 |
| tb-set-endnodeshaping | 77 |
| tb-set-noshaping | 78 |
| tb-use-endnodeshaping | 78 |
| tb-force-endnodeshaping | 79 |
| tb-set-multiplexed | 79 |
| tb-set-vlink-emulation | 80 |
| Misc. Commands..... | 80 |
| tb-fix-node | 80 |
| tb-fix-interface | 81 |
| tb-set-uselatestwadata | 81 |
| tb-set-wasolver-weights | 81 |
| tb-set-node-failure-action | 82 |
| Understanding Swapping (Node Use Policies)..... | 83 |
| What are NCL Testbed's use policies?..... | 83 |
| What is "active use"?..... | 83 |

| | |
|--|-----|
| When is an experiment considered idle? | 83 |
| What is "swapping"? | 84 |
| What is an "Idle-Swap"? | 84 |
| How long is too long for a node to be idle? | 85 |
| What is "node state"? | 86 |
| I just received an email asking me to swap or terminate my experiment. | 86 |
| Someone swapped my experiment! | 87 |
| What is "Max duration"? | 87 |
| Accessing testbed nodes using SSH | 88 |
| Uploading files to NCL..... | 88 |
| An Example Session with Windows and Putty | 88 |
| Use MobaXterm on Windows..... | 91 |
| Tips and Tricks | 91 |
| Listing your nodes from the command line | 91 |
| SSH port forwarding | 92 |
| SSH port forwarding with Putty | 92 |
| Uploading your SSH key from OS X | 96 |
| OpenSSH Configuration for Directly Logging into testbed nodes | 97 |
| Accelerating Multiple Connections using OpenSSH Connection Multiplexing | 98 |
| Node Types | 99 |
| Operating System Images | 100 |
| Supported OS Images as of 06/30/2018 | 100 |
| Creating Custom Operating System Images | 100 |
| What is an Operating System ID (OSID) versus an Image ID? | 100 |
| Standard Testbed Images..... | 101 |
| Custom OS Images..... | 101 |
| Creating Your Custom Image..... | 101 |
| Hints When Making New OS Images..... | 103 |
| Guidelines for Students | 104 |
| Student Introduction to NCL Testbed..... | 104 |
| What is NCL Testbed? | 104 |
| How does it work? | 105 |
| How do I get a NCL Testbed login? | 105 |

| | |
|--|------------|
| Using NCL Testbed..... | 106 |
| How do I start an exercise? | 106 |
| How do I work on my exercise? | 106 |
| How do I access my experiment? | 106 |
| Things to keep in mind | 107 |
| Saving and securing your files on NCL Testbed | 107 |
| Swap out -- DON'T "terminate"!..... | 107 |
| Submitting your work to your instructor | 108 |

What is NCL Testbed?

NCL Testbed is a state-of-the-art scientific computing facility for cyber-security researchers engaged in research, development, discovery, experimentation, and testing of innovative cyber-security technology. To date, NCL Testbed-based projects have included behavior analysis and defensive technologies including DDoS attacks, worm and botnet attacks, encryption, pattern detection, and intrusion-tolerant storage protocols.

Important Links

- [Testbed](#) - this is the web interface to NCL Testbed (requires [registration](#))
- Support – please email to NCL Support (support@ncl.sg).

NCL Testbed Glossary

- **boss network (`boss.ncl.sg`)**
 - The main testbed server that runs NCL Testbed. Users are not allowed to log directly into it.
- **NCL web interface (`ncl.sg`)**
 - The browser-based web portal for starting and defining experiments in NCL Testbed.
- **experiment**
 - Work in NCL Testbed is organized by experiments within project.
- **image IDs**
 - A descriptor for a disk image. This can be an image of an entire disk, a partition of a disk, or a set of partitions of a disk.
- **image**
 - Refers to a disk image.
- **link tracing/monitoring**
 - The ability to follow the path of a link or LAN in a NCL Testbed experiment.
- **nodes**
 - A node simply means any computer allocated to an experiment.
- **NS (Network Simulators) files syntax**

- Used to describe topologies in network experiments. NCL Testbed-specific information may be found in the [Core Guide](#) and further documentation is available at <http://www.isi.edu/nsnam/ns/>.

- **project / team**

- Each group of users using NCL Testbed is grouped into 'projects' or 'teams', identified by a project ID (PID).

- **swapin (or start an experiment)**

- The process where NCL Testbed allocates resources for your experiment and runs it.

- **swapout (or stop an experiment)**

- The process where NCL Testbed frees up the resources that were being used for your experiment. It is very important to do so when you are no longer using your experiment so that these resources are available for other experiments.

- **topology**

- A description of the various elements of a computer network. In NCL Testbed, your experiment requires a topology in NS syntax that describes the links, nodes, etc of your experiment.

- **users network (`users.ncl.sg`)**

- NCL Testbed's file server that serves as a shell host for testbed users.

Registering to use NCL Testbed

Who may apply for a NCL project or request a user account?

Researchers from academia, government, and industry -- as well as educators from academic institutions -- may apply for a NCL Testbed project account. A student must have their professor or appropriate faculty member apply for an account, and once it is activated the student can then apply for membership to that project.

To Register

NCL accounts are grouped by projects, therefore the project leader or PI must first request a project, then other users apply for membership to that project.

Note

If you already have a NCL Testbed account, please **login** first. This will help streamline the process.

Requesting a New Project

- **If you are a PI, project leader or instructor** who wants to request a new project on NCL Testbed, fill out the [Registration Form](#) and choose “**Apply to create a new team**”. No other users (such as students) should apply for a new team/project.
 - You will be asked a number of questions about your project and how you intend to use NCL. Please be detailed, especially with respect to any possible risks from your experiment. A NCL staff member may contact you to discuss or clarify any potential issues.
 - The project leader is responsible for ensuring that the project adheres to the Project Plan included in the application form.
 - Instructors should indicate this project is for educational purposes. Once the project is created, you will receive further instructions including how to create accounts for your students.
- Upon submission, your application must be **approved by the NCL Executive Committee**; this generally takes a few days. They may contact you and ask for clarification.
- You will receive an **email notification upon approval** and your user account will be active. You may then [log in](#) with the username and password you entered on the form.
- If you are curious about the progress on your application, [you may contact us](#).

Getting a User Account to Join an Existing Project

- If you are a team member who needs to join an existing project, fill up the [Registration Form](#) and choose “**Apply to join an existing team**”.
- The project leader will be informed via email of your request and will be required to log in to approve your account.

Note to Students

If you are a student who wants to use NCL Testbed for your own research:

- Your **faculty sponsor** must first fill out the [Registration Form](#). **A student may not create their own project.**
- Once this has been done, the sponsor will instruct you to apply for project membership, which will create your NCL Testbed account. You must obtain the name of the NCL Testbed project from your sponsor to correctly fill out the form.

If you are a student taking a class that uses NCL Testbed:

- You do not need to take any action – your instructor will create accounts and assign them to students.

If you need assistance

If you run into trouble, please [contact us](#) and we will be happy to assist.

Getting Help with NCL

Summary: take *all* of the details about your problem (project/experiment IDs, logs, error output, etc) and contact our support team.

What to do first if you are having trouble?

- Read the [FAQ page](#) to see if your question has already been answered.
- You can also search the documentation for any existing information/guide.

Frequently Asked questions

Why can't I log in?

Too many failed attempts to log into the web interface will result in your account being locked. You will get a message saying that your account has been frozen if you trigger it. If you are a student, please contact your TA. Otherwise, please [contact us](#).

Also: - You must use your **actual account name**, not an email address, to log into `users.ncl.sg`. - Too many failed attempts to log into `users.ncl.sg` will result in an **IP address ban**. We automatically whitelist all IP addresses that have successfully logged into the [web interface](#) and this list is synchronized every 15 minutes. So if you find yourself banned from connecting to `users.ncl.sg` please log into the web interface and then wait 15 minutes.

How do I copy files from my workstation to a node in an experiment?

Your home directory from `users` is available on the nodes in your experiment. Copy your files to `users.ncl.sg` using `scp` or `sftp` to make them available on your nodes.

How can I copy files from a node in the testbed to my workstation?

The reverse of the previous question: copy the files you want to your home directory, then download them from `users.ncl.sg` using `scp` or `sftp`.

Why can't I copy files to (or create files on) my home directory or project directory `/proj/YourProjName` (disk quota limit exceeded)?

There is a disk quota limit on how many files you can store on your home or project directory. The current limit is 25GB. If you have reached that limit, you need to clear files in your home directory or project directory, or move files to your nodes' local storage ([I need more disk space on my node](#)).

How can I install software on my nodes?

The currently supported operating system images (see the [List for currently supported OS images](#)) have access to full package repositories on a local mirror. Depending on your OS you may use `yum`, `apt-get`, or `pkg_add` to install software that has been pre-packaged for each OS.

If there is no package for the software you wish to install, you may install from source. Copy the source tarball to the testbed (see [How do I copy files from my workstation to a node in an experiment?](#) or use `wget` or `curl` on `users`), then follow the package's installation instructions.

While we will do everything we can to assist any issues you face, we do not have the resources to help individual users install software.

I try to swap in and get the error: Admission Control: \$project/\$experiment has too many nodes allocated!

If you are a class user: the maximum number of nodes that can be allocated for a class is limited. Wait for some of your classmates to free up resources before trying to swap in again.

You are less likely to encounter this during non-peak hours (late night and early morning) and when deadlines are distant.

If you are NOT a class user: please [contact us](#), because something is broken.

I try to swap out and get the error: /usr/testbed/bin/nfree: Please cleanup the previous errors.

This may not be so frequent an error, but may arise when the experiment deliberately brings an interface down without bringing it back up prior to swap out. Try scheduling the link back up before the end of experiment.

Your site claims that my new password is in the dictionary. I checked the dictionary and 'qwerty1234' is not in it.

Please see our [Passwords page](#).

Password Guidelines

We are a computer security testbed, so **please use a strong password**.

You may be reading this because you were told that your new password, 'qwerty1234', is in the dictionary. We do not mean the Oxford English Dictionary here. What we use is a large list of dictionary words *combined* with actual passwords that have been found in the wild. For example, the [RockYou hack](#) ended up revealing the unencrypted passwords of 32 million people (and about 14 million unique passwords).

Since this list is one of the go-to lists for the bad guys, we use it too. This means that many passwords that seem clever or obscure fail our test because someone else thought up the same thing.

Password tips:

- **The longer your password, the less likely it is to be in the dictionary.**
- **Try combining multiple words mixed with numbers and symbols.**

If you are interested in learning more about password security and cracking, this arstechnica article is a pretty good introduction: [Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331"](#).

NCL Testbed Usage Policy

Be a good citizen

NCL is a shared resource. We expect users to be good citizens by not abusing or wasting the resources we make available to them. We ask that you:

- **Read the [Core tutorial](#)** and give us feedback!
- **Do not share accounts.** We will close accounts that we suspect to be shared.
- **Use good passwords.** We are a computer security testbed, so please use a strong password. For more details, see our [Passwords page](#).
- **Swap out / stop your experiment when it is finished.** People forget to do this all the time. We do have an idle detection mechanism, but it is not perfect and may keep thinking your experiment is active long since you have collected your results, published your paper, and achieved tenure. Please log back in and **free up your nodes so that other researchers can use them**.
- **Do not abuse the no idle-swap feature.** Only turn off idle swap if there is a true need. Take the time to script the setup process for your experiment or create custom disk images.
- **Make good use of disk space.** Our goal at NCL is to assist you in running experiments. We are happy to provide you with all the storage necessary to accomplish your experimental goals. Also, we maintain nightly backups going back a few weeks. However, we are not a substitute for personal or institutional storage.
 - **We are not a backup service.** Please keep your important files backed up offsite at your institution. Our main machine room is technically subject to earthquakes, tsunamis, and liquefaction. Take a look at *rsync*.
 - **We are not an archive.** Please clean up or move offsite any large log files and traces after your experiment is done and paper is published. This makes sure that storage is available to researchers who are actively using the testbed.
 - **Keep things tidy.** Remove unused custom operating system images. Experts tell us that 22% of custom operating system images are never used even once. Why keep that around on disk so that it is backed up day after day? It slows down our backups and crash recovery disk checks, and takes resources away from other researchers.

- **If you do need more space, just contact us.** Quota limits and housekeeping requests allows us to have extra space available to allocate to you when you need it.
- **Let us know if you are done with your project.** - We'll clean it up for you!
- **Talk to us if you need something - we are here to help you.** We often can provide useful suggestions about running experiments on the testbed.

First come, first served. Sort of...

- **Do reserve your nodes as early as possible.** We are always willing to work with users to help them acquire nodes they need on time, and then help them hold those nodes as long as they need them. Sometimes this involves blocking the nodes out for a period of time. Other times this involves swapping the experiment in the night before the demo, when most nodes are free. [Please submit an order form](#) if you need a reservation and we will work with you.
- **We may swap out idle experiments.** We audit the testbed for idle experiments. When the testbed is close to full we will ping the users with long-running, idle experiments and ask if they can be swapped out. If no response is received within 8 hours, we will swap out the experiment. Note that you may prevent this by promptly replying to our email and telling us you need the nodes.
- **Very rarely, we may restrict use to a part of the testbed.** There are times when a large-scale demonstration requires us to block off a part of our testbed. At these times, you will see reduced node availability but your swapped in experiments will not be disturbed. We will inform you about these planned events at least two weeks ahead, via news items on NCL Testbed's web page.
- **Watch out for downtimes.** We have regular weekly downtimes where we reserve the right to perform service-interrupting work on the testbed (most weeks we sail right through without any noticeable interruption of service) and sometimes special downtimes are required. We usually give a few days' notice before the special downtimes. Keep an eye on the NCL Testbed Page for notices.

Privacy

NCL is a resource shared by users around the world. While we do our best to keep experiments separated from each other, we cannot provide any guarantee of privacy between projects. If you are concerned about privacy, please make sure you understand how to use UNIX permissions and encrypt your files when they are stored on our main file server. Feel free to contact us if you have any questions.

Usage is also governed by the National University of Singapore's [Privacy Policy](#).

Core Quickstart

This page describes basic information about NCL Testbed and its core functionality.

What is NCL Testbed?

The NCL Testbed is a nationally shared testbed that provides computing resources, repeatable and controllable experimentation environments, as well as application services. The testbed includes a cluster of 200 nodes that provides a wide range of provisioning mechanisms, security data and security services. NCL aims to provide a platform that fosters and encourages collaboration among researchers in academia, government bodies and the industry.

The software stack that manages the NCL testbed is based on [DETERLab](#), which is a security and education-enhanced version of [Emulab](#). NCL is also collaborating with the DETERLab team on [testbed research](#).

The NCL lab is funded by the National Research Foundation (NRF) since November 2015.

NCL Testbed (like Emulab) offers user accounts with assorted permissions associated with different experiment groups. Each group can have its own pre-configured experimental environments running on Linux, BSD, or other operating systems. Users running NCL Testbed experiments have full control of real hardware and networks running preconfigured software packages.

How does it work?

The software running NCL Testbed loads operating system images (low level disk copies) onto free nodes in the testbed, and then reconfigures programmable switches to create VLANs with the newly-imaged nodes connected according to the topology specified by the experiment creator.

After the system is fully imaged and configured, NCL Testbed executes specified scripts, unpacks tarballs, and/or installs RPM files according to the experiment's configuration. The end result is a live network of real machines, accessible via the Internet.

Work in NCL Testbed is based on projects that include individual experiments and is accomplished either via the browser-based web interface ([ncl.sg](#)) or via commandline on the NCL Testbed nodes.

To access NCL Testbed, you need to create an account, which provides credentials for accessing both the web interface and nodes.

How do I get a NCL Testbed account?

You may obtain a NCL Testbed account by either creating a new team (if you are a PI or instructor) or joining an existing team (if you are a project member or a student).

If you are the project investigator or instructor, you must create a team and invite your team members or students to join.

If you are the member of a team using NCL Testbed, your team leader will invite you to join the appropriate NCL Testbed team.

If you are a student, you may not create a team. Your instructor must create the team and, once approved, will give you information for joining the team.

See Getting Started for more information.

How do I use NCL Testbed?

In general, once you have a NCL Testbed account, you follow these steps. The NCL Testbed Core Guide will walk you through a basic tutorial of these steps.

1. Design the topology

Every experiment in NCL Testbed is based on a network topology file written in NS format and saved on the users node. The following is a very basic example:

```
# This is a simple ns script. Comments start with #.  
set ns [new Simulator]  
source tb_compat.tcl  
  
set nodeA [$ns node]  
set nodeB [$ns node]  
set nodeC [$ns node]  
set nodeD [$ns node]  
  
set link0 [$ns duplex-link $nodeB $nodeA 100Mb 50ms DropTail]
```

```
tb-set-link-loss $link0 0.01

set lan0 [$ns make-lan "$nodeD $nodeC $nodeB " 10Gb 0ms]

# Set the OS on a couple.
tb-set-node-os $nodeA Ubuntu1404-64-STD
tb-set-node-os $nodeC Ubuntu1604-64-STD

$ns rtproto Static

# Go!
$ns run
```

2. Create and start (allocate resources for) an experiment

Using your topology file, you start a new experiment via menu options in the NCL Testbed web interface (Please refer to [How to Create an Experiment](#)).

3. Generate traffic for your nodes

Now you can experiment and start generating traffic for your nodes. We will provide a flexible framework to pull together the software you'll need.

4. View results by accessing nodes, modify the experiment as needed.

Once your experiment has started, you now can access nodes via SSH and conduct your desired experiments in your new environment.

You may modify aspects of an experiment through the "Modify experiment" page on the web interface or by making changes to the NS file.

5. Save your work and stop your experiment (release the resources)

When you are ready to stop working on an experiment but know you will want to work on it again, save your files in specific protected directories and stop the experiment (via web interface or commandline) to release resources back to the testbed. This helps ensure there are enough resources for all NCL Testbed users.

This is just a high-level overview. Go to the [Core Guide](#) for a basic hands-on example of using NCL Testbed.

Core Guide

In this tutorial we begin with a small 3-5 node experiment, so that you will become familiar with NS syntax and the practical aspects of NCL Testbed operation. This is a good starting point for those new to NCL Testbed.

NCL Testbed Environment

Your experiment is made up of one or more machines on the internal NCL Testbed network, which is behind a firewall.

`users.ncl.sg` (or `users` for short) is the "control server" for NCL Testbed. From `users`, you can contact all your nodes, reboot them, etc. Each user has a home directory on this server and you may SSH into it with your username and password for your NCL Testbed account.

`boss.ncl.sg` (or `boss` for short) is the main testbed server that runs NCL Testbed. Users are not allowed to log directly into it.

Basic Tutorial

Getting Started

Work in NCL Testbed is organized by **experiments** within **projects**. Each project is created and managed by a leader - usually the Principal Investigator (PI) of a research project or the instructor of a class on cybersecurity. The project leader then invites members to join by providing them with the project name and sending them the link to the 'Join a Project' page.

Before you can take the following tutorial, you need an active account in a project in NCL Testbed. See [How to register](#) to make sure if you're qualified, and then follow the directions to create a project or ask to join an existing project - if you go through either process for the first time, your account is created as a result.

If you already have an account, proceed to the next step.

Step 1: Design the topology

Part of NCL Testbed's power lies in its ability to assume many different topologies; the description of such a topology is a necessary part of an experiment. Before you can start your experiment, you must model the elements of the experiment's network with a topology.

For this basic tutorial, we use the below NS file which includes a simple topology of 4 nodes. You can save the file to your local computer.

NS File:

```
# This is a simple experiment, Comments start with #.  
  
set ns [new Simulator]  
source tb_compat.tcl  
  
# Add a new node  
set nodeA [$ns node]  
set nodeB [$ns node]  
set nodeC [$ns node]  
set nodeD [$ns node]  
  
# Define the link and LAN  
set link0 [$ns duplex-link $nodeB $nodeA 100Mb 50ms DropTail]  
tb-set-link-loss $link0 0.01  
set lan0 [$ns make-lan "$nodeD $nodeC $nodeB " 10Gb 0ms]  
  
# Set node OS  
tb-set-node-os $nodeA Ubuntu1404-64-STD  
tb-set-node-os $nodeB Ubuntu1604-64-STD  
tb-set-node-os $nodeC Ubuntu1604-64-GUI  
  
$ns rtproto Static  
  
# Go!  
$ns run
```

The rest of this section describes NS format and walks you through the different parts of the sample file.

NS Format

NCL Testbed uses the "NS" ("Network Simulator") format to describe network topologies. This is substantially the same [Tcl-based format](#) used by [ns-2](#). Since NCL Testbed offers emulation, rather than simulation, these files are interpreted in a somewhat different manner than ns-2. Therefore, some ns-2 functionality may work differently than you expect, or may not be implemented at all. Please look for warnings of the form:

```
*** WARNING: Unsupported NS Statement!  
Link type BAZ, using DropTail!
```

If you feel there is useful functionality missing, please [let us know](#). Also, some [testbed-specific syntax](#) has been added, which, with the inclusion of the compatibility module `tb_compat.tcl`, will be ignored by the NS simulator. This allows the same NS file to work on both NCL Testbed and ns-2, most of the time.

NS Example

In our example, we are creating a test network which looks like the following:

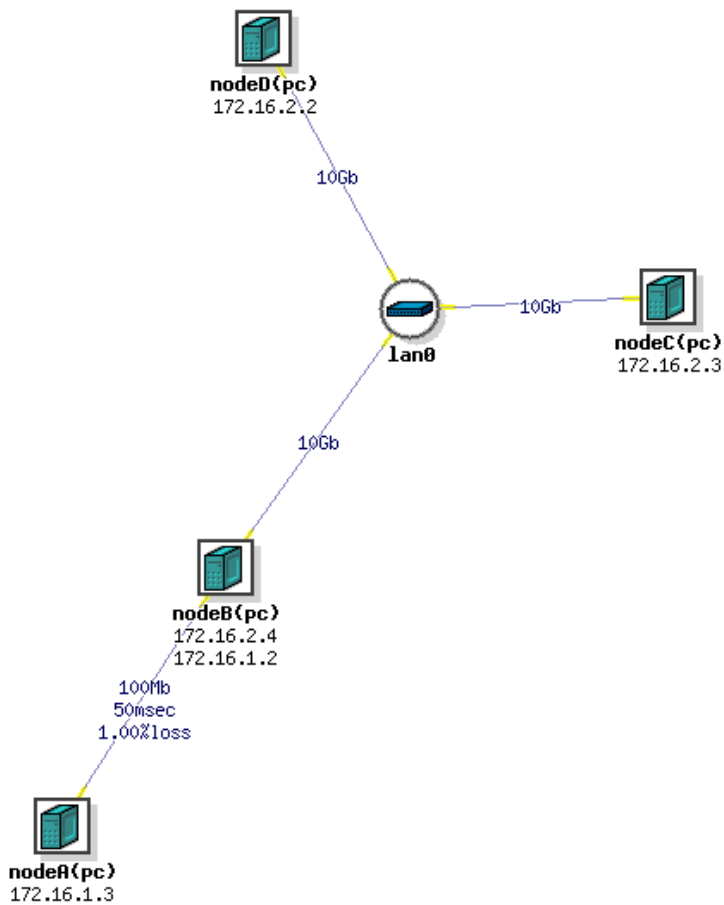


Figure 1: A is connected to B, and B to C and D with a LAN.

Here's how to describe this topology:

Declare a simulator and include a file that allows you to use the special `tb-` commands.

First off, all NS files start with a simple prologue, declaring a simulator and including a file that allows you to use the special `tb-` commands:

```
# This is a simple ns script. Comments start with #.
set ns [new Simulator]
source tb_compat.tcl
```

Define the 4 nodes in the topology.

```
set nodeA [$ns node]
set nodeB [$ns node]
```

```
set nodeC [$ns node]
set nodeD [$ns node]
```

nodeA and so on are the virtual names (*vnames*) of the nodes in your topology. When your experiment is swapped in (has allocated resources), they will be assigned to physical node names like *pc10a*, probably different ones each time.

NOTE: Avoid vnames that clash with the physical node names in the testbed. **

Define the link and the LAN that connect the nodes.

NS syntax permits you to specify the bandwidth, latency, and queue type. Note that since NS can't impose artificial losses like NCL Testbed can, we use a separate `tb-` command to add loss on a link. For our example, we will define a full speed LAN between B, C, and D, and a shaped link from node A to B.

```
set link0 [$ns duplex-link $nodeB $nodeA 100Mb 50ms DropTail]
tb-set-link-loss $link0 0.01
set lan0 [$ns make-lan "$nodeD $nodeC $nodeB " 10Gb 0ms]
```

In addition to the standard NS syntax above, a number of [extensions](#) are available in NCL Testbed that allow you to better control your experiment.

For example, you may specify what Operating System is booted on your nodes. For the versions of FreeBSD and Linux we currently support, please refer to the [Operating System Images](#) page.

Click [List of Standard OS Images](#) in the NCL Testbed web interface to see the current list of NCL Testbed-supplied operating systems. By default, our most recent Linux image is selected.

```
tb-set-node-os $nodeA Ubuntu1404-64-STD
tb-set-node-os $nodeB Ubuntu1604-64-STD
tb-set-node-os $nodeC Ubuntu1604-64-GUI
```

Enable routing.

In a topology like this, you will likely want to communicate between all the nodes, including nodes that aren't directly connected, like **A** and **C**. In order for that to happen, we must enable routing in our experiment, so **B** can route packets for the other nodes.

The typical way to do this is with Static routing. (Other options are detailed in the [Routing section below](#)).

```
$ns rtproto Static
```

End with an epilogue that instructs the simulator to start.

```
# Go!  
$ns run
```

Step 2: Create & start a new experiment

For this tutorial, we will use the web interface to create a new experiment. You could also use the [NCL Testbed Shell Commands](#).

1. Log into [NCL Testbed](#) with your account credentials.
2. Click the *EXPERIMENT* menu item, then click *Create Experiment*.
3. Click *Select Team* and choose your team. This is also known as your **project name** or Project ID (PID). This is used as an argument in many commands. Most people will be a member of just one project, and will not have a choice. If you are a member of multiple projects, be sure to select the correct project from the menu. In this example, we will refer to the project as *ncltest*.
4. Enter the *Name* field with an easily identifiable name for the experiment. The Name should be a single word (no spaces) identifier. For this tutorial, use *basic-experiment*. This is also known as your **experiment name** or Experiment ID (EID) and is used as an argument in many commands.
5. Enter the *Description* field with a brief description of the experiment.
6. In the **Network Configuration** field, copy-paste the content of [the NS file shown in the previous page](#).
7. Click *Submit* (you will be redirected to the List of Experiments page if the experiment is successfully created).
8. Click *Start this experiment* for the experiment you have just created.

After submission, NCL Testbed will begin processing your request. This process can take up to 20 minutes, depending on how large your topology is, and what other features (such as delay

nodes and bandwidth limits) you are using. While you are waiting, you may watch the starting process displayed in your web browser.

Assuming all goes well, the status of your experiment will be changed to Running, and you can view the listing of the nodes and IP address that were allocated to your experiment (Click *EXPERIMENT* menu, then click the experiment name you just created, and then click *Realization* tab).

For the NS file in this example, you should see a listing that looks similar to this:

```
Experiment: ncltest01/basic-experiment
State: active
```

Virtual Node Info:

| ID | Type | OS | Qualified Name |
|-----------------------------|------|-------------------|----------------|
| nodeA | pc | Ubuntu1404-64-STD | nodeA.basic- |
| experiment.ncltest01.ncl.sg | | | |
| nodeB | pc | Ubuntu1604-64-STD | nodeB.basic- |
| experiment.ncltest01.ncl.sg | | | |
| nodeC | pc | Ubuntu1604-64-GUI | nodeC.basic- |
| experiment.ncltest01.ncl.sg | | | |
| nodeD | pc | | nodeD.basic- |
| experiment.ncltest01.ncl.sg | | | |

Physical Node Mapping:

| ID | Type | OS | Physical |
|----------|---------|---------------------|----------|
| nodeA | SystemX | Ubuntu1404-64-STD | pc21a |
| nodeB | Intel | Ubuntu1604-64-STD | pc21f |
| nodeC | Intel | Ubuntu1604-64-GUI | pc22f |
| nodeD | Intel | Ubuntu1404-64-STD | pc4f |
| tbdelay0 | SystemX | Ubuntu1204-64-Click | pc24a |

Virtual Lan/Link Info:

| ID | Member/Proto | IP/Mask | Delay | BW (Kbs) | Loss Rate |
|------------|--------------|---------------|-------|----------|-----------|
| lan0 | nodeB:1 | 172.16.2.4 | 0.00 | 10000000 | |
| 0.00000000 | ethernet | 255.255.255.0 | 0.00 | 10000000 | |
| 0.00000000 | | | | | |
| lan0 | nodeC:0 | 172.16.2.3 | 0.00 | 10000000 | |
| 0.00000000 | ethernet | 255.255.255.0 | 0.00 | 10000000 | |
| 0.00000000 | | | | | |
| lan0 | nodeD:0 | 172.16.2.2 | 0.00 | 10000000 | |
| 0.00000000 | ethernet | 255.255.255.0 | 0.00 | 10000000 | |
| 0.00000000 | | | | | |
| link0 | nodeA:0 | 172.16.1.3 | 25.00 | 100000 | |
| 0.00501256 | | | | | |

| | | | | | |
|------------------------------|----------|---------------|-------------------|--------------|----------------|
| 0.00501256 | ethernet | 255.255.255.0 | 25.00 | 100000 | |
| link0 | nodeB:0 | 172.16.1.2 | 25.00 | 100000 | |
| 0.00501256 | ethernet | 255.255.255.0 | 25.00 | 100000 | |
| 0.00501256 | | | | | |
| Physical Lan/Link Mapping: | | | | | |
| ID | Member | IP | MAC | | NodeID |
| ----- | ----- | ----- | ----- | | ----- |
| - | | | | | |
| lan0 | nodeB:1 | 172.16.2.4 | a0:f4:79:3d:f5:38 | 4/1 <-> 1/41 | pc21f |
| HPClus7Expt | | | | | |
| lan0 | nodeC:0 | 172.16.2.3 | a0:f4:79:7e:b0:93 | 4/1 <-> 1/43 | pc22f |
| HPClus7Expt | | | | | |
| lan0 | nodeD:0 | 172.16.2.2 | a0:f4:79:8e:c7:0c | 5/1 <-> 1/8 | pc4f |
| HPClus7Expt | | | | | |
| link0 | nodeA:0 | 172.16.1.3 | 40:f2:e9:1e:28:c4 | 1/1 <-> 1/42 | pc21a |
| HPClus2Expt | | | | | |
| link0 | nodeB:0 | 172.16.1.2 | a0:f4:79:3d:f5:39 | 5/1 <-> 1/42 | pc21f |
| HPClus7Expt | | | | | |
| Virtual Queue Info: | | | | | |
| ID | Member | Q Limit | Type | | |
| weight/min_th/max_th/linterm | | | | | |
| ----- | ----- | ----- | ----- | | ----- |
| -- | | | | | |
| lan0 | nodeB:1 | 100 slots | Tail | 0/0/0/0 | |
| lan0 | nodeC:0 | 100 slots | Tail | 0/0/0/0 | |
| lan0 | nodeD:0 | 100 slots | Tail | 0/0/0/0 | |
| link0 | nodeA:0 | 100 slots | Tail | 0/0/0/0 | |
| link0 | nodeB:0 | 100 slots | Tail | 0/0/0/0 | |
| Physical Delay Info: | | | | | |
| ID | Member | Delay Node | Delay | BW (Kbs) | PLR Pipe |
| ----- | ----- | ----- | ----- | ----- | ----- |
| -- | | | | | |
| link0 | nodeA | tbdelay0 | 50.00 | 100000 | 0.00999999 110 |
| link0 | nodeB | tbdelay0 | 50.00 | 100000 | 0.00999999 120 |
| Physical Queue Info: | | | | | |
| ID | Member | Q Limit | Type | | |
| weight/min_th/max_th/linterm | | | | | |
| ----- | ----- | ----- | ----- | | ----- |
| -- | | | | | |
| link0 | nodeA | 100 slots | Tail | 0/0/0/0 | |
| link0 | nodeB | 100 slots | Tail | 0/0/0/0 | |
| link0 | nodeA | 100 slots | Tail | 0/0/0/0 | |
| link0 | nodeB | 100 slots | Tail | 0/0/0/0 | |
| Delay Node Switch Info: | | | | | |
| ID | Member | Delay Node | Card/Port | Switch | Card/Port |

```

-----
link0          nodeA          tbdelay0      0/1 eth0  HPCLus2Expt 1/47
link0          nodeB          tbdelay0      1/1 eth1  HPCLus2Expt 1/48

Event Groups:
Group Name     Members
-----
--
link0-tracemon link0-nodeB-tracemon,link0-nodeA-tracemon
__all_tracemon link0-nodeB-tracemon,link0-nodeA-tracemon,lan0-nodeD-
tracemon,lan0-nodeC-tracemon,lan0-nodeB-tracemon
__all_lans      lan0,link0
__all_program-agents __tbdelay0_program-agent
lan0-tracemon   lan0-nodeB-tracemon,lan0-nodeC-tracemon,lan0-nodeD-tracemon

Event Summary:
-----
Event count:    1
First event:    0.000 seconds
Last event:     0.000 seconds

```

Here is a breakdown of the results:

- A single delay node was allocated and inserted into the link between *nodeA* and *nodeB*. This link is invisible from your perspective, except for the fact that it adds latency, error, or reduced bandwidth. However, the information for the delay links are included so that you can modify the delay parameters after the experiment has been created (Note that you cannot convert a non-shaped link into a shaped link; you can only modify the traffic shaping parameters of a link that is already being shaped).
- Delays of less than 2ms (per trip) are too small to be accurately modeled at this time, and will be silently ignored. A delay of 0ms can be used to indicate that you do not want added delay; the two interfaces will be "directly" connected to each other.
- Each link in the *Virtual Lan/Link* section has its delay, etc., split between two entries. One is for traffic coming into the link from the node, and the other is for traffic leaving the link to the node. In the case of links, the four entries often get optimized to two entries in a *Physical Lan/Link* section.
- The names in the *Qualified Name* column refer to the control network interfaces for each of your allocated nodes. These names are added to the NCL Testbed nameserver map on the fly, and are immediately available for you to use so that you do not have to worry about the actual physical node names that were chosen. In the names listed above, **ncltest01** is the

name of the project that you chose to work in, and **basic-experiment** is the name of the experiment that you provided on the *Create Experiment* page.

- Please don't use the *Qualified Name* from within nodes in your experiment, since it will contact them over the control network, bypassing the link shaping we configured.

Step 3: Access nodes in your lab environment

To access your experimental nodes, you'll need to first [SSH](#) into `users.ncl.sg` using your NCL Testbed username and password.

Once you log in to `users`, you'll need to SSH again to your actual experimental nodes. Since your nodes addresses may change every time you swap them in, it's best to SSH to the permanent network names of the nodes.

As we mentioned in the previous step, the Qualified Names are included in the output after the experiment is started. Here is another way to find them after swap-in:

a. **Navigate to the List of Experiments page on the web interface.**

b. **Click *View Realization* for the experiment you just started.** * Your nodes' network names are listed under the heading *Qualified Name*. For example, `nodeA.basic-experiment.ncltest01.ncl.sg`. * You should familiarize yourself with the information available on this page, but for now we just need to know the long DNS qualified name(s) node(s) you just started.

c. **SSH from `users` to your experimental nodes by running a command with the following syntax:**

```
ssh nodeX.ExperimentName.ProjectName.ncl.sg
```

```
* You will not need to re-authenticate.
```

```
* You may need to wait a few more minutes. Once NCL Testbed is finished setting up the experiment, the nodes still need a minute or two to boot and complete their configuration. If you get a message about "server configuration" when you try to log in, wait a few minutes and try again.
```

d. If you need to create new users on your experimental nodes, you may log in as them by running the following from the experimental node:

```
ssh newuser@node1.basicExp.ProjectName.ncl.sg Or  
ssh newuser@localhost
```

Step 4: View results and modify the experiment

You can visualize the experiment by going to your *Experiments* page and clicking the *View Visualization* button for the experiment. From this page you can also change the NS file by clicking on the *Modify this experiment* button (you need to stop the experiment first before you can modify it).

An alternative method is to log into `users.ncl.sg` and use the [delay_config](#) program. This program requires that you know the symbolic names of the individual links. This information is available on the experiment page.

Step 5: Configure and run your experiment.

Once you have all link modifications to your liking, you now need to install any additional tools you need (tools not included in the OS images you chose in Step 1), configure your tools and coordinate these tools to create events in your experiment.

For simple experiments, installation, configuration and triggering events can be done by hand or through small scripts. To accomplish this, log into your machines (see Step 3), perform the OS-specific steps needed to install and configure your tools, and run these tools by hand or through scripts, such as shell scripts or remote scripts such as Fabric-based scripts <http://www.fabfile.org>.

Step 6: Save your work and swap out (release resources)

When you are done working with your nodes, it is best practice to save your work and stop the experiment so that other users have access to the physical machines.

Saving and securing your files on NCL Testbed

Every user on NCL Testbed has a home directory (`/users/xxx`) on `users.ncl.sg` which is mounted via NFS to experimental nodes. This means that anything you place in your home directory on one experimental node (or the `users` machine) is visible in your home directory on your other experimental nodes. Your home directory is private and will not be overwritten, so

you may save your work in that directory. **However, everything else on experimental nodes is permanently lost when an experiment is stopped.**

Remember: Make sure you save your work in your home directory before stopping your experiment!

Another place you may save your files would be `/proj/YourProject`. This directory is also NFS-mounted to all experimental nodes, so the same rules apply about writing to it a lot, as for your home directory. It is shared by all members of your project/class.

NOTE: There is a disk quota limit on how many files you can store on your home or project directory. The current limit is 25GB. If you have reached that limit, you need to clear files in your home directory or project directory, or move files to your nodes' local storage (see [I need more disk space on my node](#)).

Again, on NCL Testbed, files ARE NOT SAVED between restarts. Additionally, experiments may be forcibly stopped after a certain number of idle hours (or some maximum amount of time).

You must manually save copies of any files you want to keep in your home directory. Any files left elsewhere on the experimental nodes will be erased and lost forever. This means that if you want to store progress for a lab and come back to it later, you will need to put it in your home directory before stopping the experiment.

Swap Out vs Terminate

When to Swap Out When you are done with your experiment for the time being, make sure you save your work into an appropriate location and then swap out your experiment. Swapping out is the equivalent of temporarily stopping the experiment and relinquishing the testbed resources. Swapping out is what you want to do when you are taking a break from the work, but coming back later.

To do this, click *Stop this experiment* button on the Experiments page. This allows the resources to be de-allocated so that someone else can use them.

When to Terminate When you are completely finished with your experiment and have no intention of running it again, use the *Remove Experiment* button on the Experiments page. Be

careful: **termination will erase the experiment** and you won't be able to swap it back in without recreating it. NCL Testbed will then tear down your experiment, and send you an email message when the process is complete. At this point you are allowed to reuse the experiment name (say, if you wanted to create a similar experiment with different parameters).

Terminating says "I won't need this experiment ever again." Just remember to Swap In/Out, and never "Terminate" unless you're sure you're completely done with the experiment. If you do end up terminating an experiment, you can always go back and recreate it.

Scheduling experiment swapout/termination

If you expect that your experiment should run for a set period of time, but you will not be around to terminate or swap the experiment out, then you should use the scheduled swapout/termination feature. This allows you to specify a maximum running time in your NS file so that you will not hold scarce resources when you are offline. To schedule a swapout or termination in your NS file:

```
$ns at 2000.0 "$ns terminate"
```

or

```
$ns at 2000.0 "$ns swapout"
```

This will cause your experiment to either be terminated or swapped out after 2000 seconds of wallclock time.

Why can't I log in to NCL Testbed?

NCL Testbed has an automatic blacklist mechanism. If you enter the wrong username and password combination too many times, your account will no longer be accessible from your current IP address.

If you think that this has happened to you, try logging in from another address (if you know how), or create an issue (see [Getting Help](#)), which will relay the request to the *testbed-ops* group that this specific blacklist entry should be erased.

Installing RPMs automatically

The NCL Testbed NS extension `tb-set-node-rpms` allows you to specify a (space-separated) list of RPMs to install on each of your nodes when it boots:

```
tb-set-node-rpms $nodeA /proj/myproj/rpms/silly-freebsd.rpm
tb-set-node-rpms $nodeB /proj/myproj/rpms/silly-linux.rpm
tb-set-node-rpms $nodeC /proj/myproj/rpms/silly-windows.rpm
```

The above NS code says to install the `silly-freebsd.rpm` file on `nodeA`, the `silly-linux.rpm` on `nodeB`, and the `silly-windows.rpm` on `nodeC`. RPMs are installed as root, and must reside in either the project's `/proj` directory, or if the experiment has been created in a subgroup, in the `/groups` directory. You may not place your RPMs in your home directory.

Installing TAR files automatically

The NCL Testbed NS extension `tb-set-node-tarfiles` allows you to specify a set of tarfiles to install on each of your nodes when it boots.

While similar to the `tb-set-node-rpms` command, the format of this command is slightly different in that you must specify a directory in which to unpack the tar file. This avoids problems with having to specify absolute pathnames in your tarfile, which many modern tar programs balk at.

```
tb-set-node-tarfiles $nodeA /usr/site /proj/projectName/tarfiles/silly.tar.gz
```

The above NS code says to install the `silly.tar.gz` tar file on `nodeA` from the working directory `/usr/site` when the node first boots. The tarfile must reside in either the project's `/proj` directory, or if the experiment has been created in a subgroup, in the `/groups` directory. You may not place your tarfiles in your home directory. You may specify as many tarfiles as you wish, as long as each one is preceded by the directory it should be unpacked in, all separated by spaces.

Starting your application automatically

You may start your application automatically when your nodes boot for the first time (when an experiment is started or swapped in) by using the `tb-set-node-startcmd` NS extension. The argument is a command string (pathname of a script or program, plus arguments) that is run as the `UID` of the experiment creator, after the node has reached multiuser mode.

The command is invoked using `/bin/csh`, and the working directory is undefined (your script should `cd` to the directory you need). You can specify the same program for each node, or a different program. For example:


```
tb-set-node-startcmd $nodeA "/proj/projectName/runme.nodeA"  
tb-set-node-startcmd $nodeB "/proj/projectName/runme.nodeB"
```

will run `/proj/projectName/runme.nodeA` on nodeA and `/proj/projectName/runme.nodeB` on nodeB. The programs must reside on the node's local filesystem, or in a directory that can be reached via NFS. This is either the project's `/proj` directory, in the `/groups` directory if the experiment has been created in a subgroup, or a project member's home directory in `/users`.

If you need to see the output of your command, be sure to redirect the output into a file. You may place the file on the local node, or in one of the NFS mounted directories mentioned above. For example:

```
tb-set-node-startcmd $nodeB "/proj/myproj/runme >& /tmp/foo.log"
```

Note that the syntax and function of `/bin/csh` differs from other shells (including bash), specifically in redirection syntax. Be sure to use `csh` syntax or your start command will fail silently.

Notifying the start program when all other nodes have started

It is often necessary for your start program to determine when all of the other nodes in the experiment have started, and are ready to proceed. Sometimes called a *barrier*, this allows programs to wait at a specific point, and then all proceed at once. NCL Testbed provides a simple form of this mechanism using a synchronization server that runs on a node of your choice.

Specify the node in your NS file:

```
tb-set-sync-server $nodeB
```

When nodeB boots, the synchronization server will automatically start. Your software can then synchronize using the `emulab-sync` program that is installed on your nodes. For example, your node start command might look like this:

```
#!/bin/sh  
if [ "$1" = "master" ]; then  
    /usr/testbed/bin/emulab-sync -i 4
```

```
else
    /usr/testbed/bin/emulab-sync fi /usr/site/bin/dosilly
```

In this example, there are five nodes in the experiment, one of which must be configured to operate as the master, initializing the barrier to the number of clients (four in the above example) that are expected to rendezvous at the barrier. The master will by default wait for all of the clients to reach the barrier. Each client of the barrier also waits until all of the clients have reached the barrier (and of course, until the master initializes the barrier to the proper count). Any number of clients may be specified (any subset of nodes in your experiment can wait). If the master does not need to wait for the clients, you may use the *async* option which releases the master immediately:

```
/usr/testbed/bin/emulab-sync -a -i 4
```

You may also specify the *name* of the barrier.

```
/usr/testbed/bin/emulab-sync -a -i 4 -n mybarrier
```

This allows multiple barriers to be in use at the same time. Scripts on nodeA and nodeB can be waiting on a barrier named "foo" while (other) scripts on nodeA and nodeC can be waiting on a barrier named "bar." You may reuse an existing barrier (including the default barrier) once it has been released (all clients arrived and woken up).

Setting up IP routing between nodes

As NCL strives to make all aspects of the network controllable by the user, we do not attempt to impose any IP routing architecture or protocol by default. However, many users are more interested in end-to-end aspects and don't want to be bothered with setting up routes. For those users we provide an option to automatically set up routes on nodes which run one of our provided FreeBSD or Linux disk images.

You can use the NS `rtproto` syntax in your NS file to enable routing:

```
$ns rtproto protocolOption
```

where the `protocolOption` is limited to one of *Session*, *Static*, *Static-old*, or *Manual*.

- **Session** routing provides fully automated routing support, and is implemented by enabling `gated` running of the OSPF protocol on all nodes in the experiment.

- **Static** routing also provides automatic routing support, but rather than computing the routes dynamically, the routes are precomputed by a distributed route computation algorithm running in parallel on the experiment nodes.
- **Static-old** specifies use of the older centralized route computation algorithm, precomputing the nodes when the experiment is created, and then loading them onto each node when it boots.
- **Manual** routing allows you to explicitly specify per-node routing information in the NS file. To do this, use the `Manual` routing option to `rtproto`, followed by a list of routes using the `add-route` command:

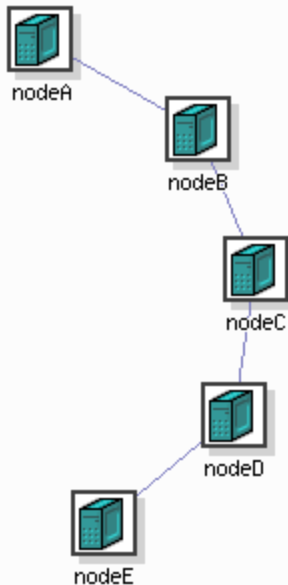
```
$node add-route $dst $nexthop
```

where the `dst` can be either a node, a link, or a LAN. For example:

```
$client add-route $server $router
$client add-route [$ns link $server $router] $router
$client add-route $serverlan $router
```

Note that you would need a separate `add-route` command to establish a route for the reverse direction; thus allowing you to specify differing forward and reverse routes if so desired. These statements are converted into appropriate `route(8)` commands on your experimental nodes when they boot.

In the above examples, the first form says to set up a manual route between `$client` and `$server`, using `$router` as the next hop; `$client` and `$router` should be directly connected, and the interface on `$server` should be unambiguous; either directly connected to the router, or an edge node that has just a single interface.



If the destination has multiple interfaces configured, and it is not connected directly to the nexthop, the interface that you are intending to route to is ambiguous. In the topology shown to the right, `$nodeD` has two interfaces configured. If you attempted to set up a route like this:

```
$nodeA add-route $nodeD $nodeB
```

you would receive an error since NCL Testbed staff would not easily be able to determine which of the two links on `$nodeD` you are referring to. Fortunately, there is an easy solution. Instead of a node, specify the link directly:

```
$nodeA add-route [$ns link $nodeD $nodeC] $nodeB
```

This tells us exactly which link you mean, enabling us to convert that information into a proper `route` command on `$nodeA`.

The last form of the `add-route` command is used when adding a route to an entire LAN. It would be tedious and error prone to specify a route to each node in a LAN by hand. Instead, just route to the entire network:

```
set clientlan [$ns make-lan "$nodeE $nodeF $nodeG" 10Gb 0ms]
$nodeA add-route $clientlan $nodeB
```

In general, it is still best practice to use either *Session* or *Static* routing for all but small, simple topologies. Explicitly setting up all the routes in even a moderately-sized experiment is

extremely error prone. Consider this: a recently created experiment with 17 nodes and 10 subnets **required 140 hand-created routes in the NS file.**

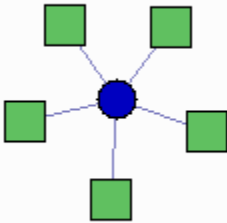
Two final, cautionary notes on routing: * The default route *must* be set to use the control network interface. You might be tempted to set the default route on your nodes to reduce the number of explicit routes used. **Please avoid this.** That would prevent nodes from contacting the outside world, i.e., you. * If you use your own routing daemon, you must avoid using the control network interface in the configuration. Since every node in the testbed is directly connected to the control network LAN, a naive routing daemon configuration will discover that any node is just one hop away, via the control network, from any other node and *all* inter-node traffic will be routed via that interface.

Sample Topologies

The following are various topologies you can use to experiment with NCL Testbed.

Toy topologies

LAN



```
set ns [new Simulator]
source tb_compat.tcl

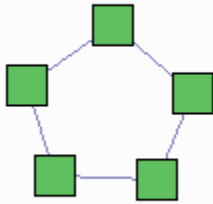
# Change this to a number of nodes you want
set NODES 5

set lanstr ""
for {set i 0} {$i < $NODES} {incr i} {
    set node($i) [$ns node]
    append lanstr "$node($i) "
}

# Change the BW and delay if you want
set lan0 [$ns make-lan "$lanstr" 10Gb 0ms]

$ns rtproto Static
$ns run
```

Ring



```
set ns [new Simulator]
source tb_compat.tcl

# Change this to a number of nodes you want
set NODES 5

set node(0) [$ns node]
for {set i 1} {$i < $NODES} {incr i} {
    set node($i) [$ns node]
    set lastindex [expr $i-1]

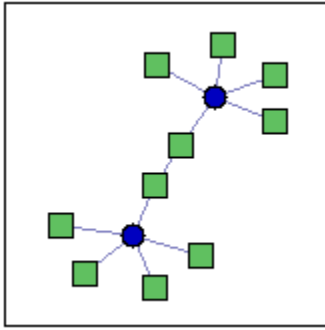
    # Change BW and delay if you want
    set Link$i [$ns duplex-link $node($i) $node($lastindex) 10Gb 0ms
DropTail]
}

set lastindex [expr $i-1]

# Change BW and delay if you want
set Link$i [$ns duplex-link $node(0) $node($lastindex) 10Gb 0ms DropTail]

$ns rtproto Static
$ns run
```

Dumbbell



```
set ns [new Simulator]
source tb_compat.tcl

# Add nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]

# Change BW and delay if you want
set lan0 [$ns make-lan "$n0 $n1 $n2 $n3 $n4" 10Gb 0ms]
set lan1 [$ns make-lan "$n5 $n6 $n7 $n8 $n9" 10Gb 0ms]

# Change BW and delay if you want
set link0 [$ns duplex-link $n4 $n5 10Gb 0ms DropTail]

$ns rtproto Static

# Go!
$ns run
```


Using Your Nodes

Know your NCL servers

Here are the most important things to know.

- `ncl.sg` is the primary web interface for the testbed.
- `users.ncl.sg` is the host through which the testbed nodes are accessed and it is primary file server.
- `scratch` is the local package mirror for CentOS, Ubuntu, and FreeBSD.

Hostnames for your nodes

We set up names for your nodes in DNS and `/etc/hosts` files for use on the nodes in the experiment. Since our nodes have multiple interfaces (the control network, and, depending on the experiment, possibly several experimental interfaces) determining which name refers to which interface can be somewhat confusing. The rules below should help you figure this out.

- **From `users.ncl.sg`** - We set up names in the form of `node.expt.proj.ncl.sg` in DNS, so that they are visible anywhere on the Internet. This name always refers to the node's control network interface, which is the only one reachable from `users.ncl.sg`. You can use `node.expt.proj` as a shorthand.
- **On the nodes themselves** - There are three basic ways to refer to the interfaces of a node. The first is stored in DNS, and the second two are stored on the node in the `/etc/hosts` file.
 1. *Fully-qualified hostnames* - These names are the same ones visible from the outside world, and referred to by attaching the full domain name: ie. `node.expt.proj.ncl.sg`. (Note that, since we put `.ncl.sg` in the nodes' domain search paths, you can use `node.expt.proj` as a shorthand.) This name always refers to the control network.
 2. *node-link form* - You can refer to an individual experimental interface by suffixing it with the name of the link or LAN (as defined in your NS file) that it is a member of. For example, `nodeA-link0` or `server-serverLAN`. This is the preferred way to refer to experimental interfaces, since it uniquely and unambiguously identifies an interface.
 3. *Short form* - If a node is directly connected to the node you're on, you can refer to that node simply with its name (eg. `nodeA`.) Note that this differs from the fully-qualified name in that no

domain is given. We also create short names for nodes you are not directly connected to. However, if two nodes are connected with more than one interface, or there is more than one route between them, there is no guarantee that the short name has been associated with the one is on the best (ie. shortest or highest bandwidth) path - so, if there is ambiguity, we strongly suggest that you use the *node-link* form.

Note

It is a bad idea to pick virtual node names in your topology that clash with the physical node names in the testbed, such as "pc10a".

Logging into your Node

If you have selected one of the NCL-supported operating system images ([see supported images](#)), this configuration process includes: * loading fresh disk images so that each node is in a known clean state; * rebooting each node so that it is running the OS specified in the NS script; * configuring each of the network interfaces so that each one is "up" and talking to its virtual LAN (VLAN); * creating user accounts for each of the project members; * mounting the projects NFS directory in /proj so that project files are easily shared amongst all the nodes in the experiment; * creating a /etc/hosts file on each node so that you may refer to the experimental interfaces of other nodes by name instead of IP number; * configuring all of the delay parameters; * configuring the serial console lines so that project members may access the console ports from users.ncl.sg.

As this point you may log into any of the nodes in your experiment. You will need to use [Secure Shell \(ssh\)](#) to log into `users.ncl.sg` Your login name will be displayed in the Dashboard page after you have logged into the Web Interface.

Note

Although you log into the web interface using your email address instead of your login name, you must use your login name when logging into `users.ncl.sg`.

Once logged into users you can then SSH to your nodes. You should use the 'qualified name' from the nodes mapping table so that you do not form dependencies on any particular physical node. For more information on using SSH with NCL, please take a look at the [NCL SSH](#) page.

How do I install software onto my node?

Each [supported operating system](#) has packages mirrored on a host called scratch and each operating system is configured to use this system to fetch packages from. Information for specific operating systems is documented there.

How do I copy files onto my node?

Your home directory on `users` is automatically mounted via NFS on every node in your experiment. As are your project directory in `/proj` and a special filesystem called `/share`.

NOTE: There is a disk quota limit on how many files you can store on your home or project directory. The current limit is 25GB. If you have reached that limit, you need to clear files in your home directory or project directory, or move files to your nodes' local storage (see [I need more disk space on my node](#)).

I need root access!

If you need to customize the configuration, or perhaps reboot nodes, you can use the "sudo" command, located in `/usr/local/bin` on FreeBSD and `/usr/bin` Linux. Our policy is very liberal; you can customize the configuration in any way you like, provided it does not interfere with the operation of the testbed. As an example, to reboot a node that is running Ubuntu:

```
sudo reboot
```

My node is wedged!

Power cycling a node is easy since every node on the testbed is connected to a power controller. If you need to power cycle a node, log on to `users.ncl.sg` and use the "node_reboot" command:

```
node_reboot <node> [node ... ]
```

where `node` is the physical name, as listed in the node mapping table. You may provide more than one node on the command line. Be aware that you may power cycle only nodes in projects that you are member of. Also, `node_reboot` does its very best to perform a clean reboot before resorting to cycling the power to the node. This is to prevent the damage that can occur from constant power cycling over a long period of time. For this reason, `node_reboot` may delay a minute or two if it detects that the machine is still responsive to network transmission. In any event, please try to reboot your nodes first (see above). You may also reboot all the nodes

in an experiment by using the `-e` option to specify the project and experiment names. For example:

```
node_reboot -e testbed,multicast
```

will reboot all of the nodes reserved in the "multicast" experiment in the "testbed" project. This option is provided as a shorthand method for rebooting large groups of nodes.

I want to load a fresh operating system on my node

Scrogging your disk is certainly not as common, but it does happen. You can either swap your experiment out and then back in (which will allocate another group of nodes), or if you prefer you can reload the disk image yourself. You will of course lose anything you have stored on that disk; it is a good idea to store only data that can be easily recreated, or else store it in your project directory in `/proj`.

Reloading your disk with a fresh copy of an image is easy, and requires no intervention by NCL staff:

```
os_load [-i ImageName] [-p Project] <node> [node ... ]
```

If you do not specify an image name, the default image for that node type will be loaded (typically Ubuntu1404-64-STD). For testbed wide images, you do not have to specify a project. The `os_load` command will wait (not exit) until the nodes have been reloaded.

For example, to load the image 'testpc167' which is in the project 'DeterTest' onto pc167, we type:

```
users > os_load -i testpc167 -p DeterTest pc167
osload (pc167): Changing default OS to [OS 998: DeterTest,testpc167]
osload: Updating image signature.
Setting up reload for pc167 (mode: Frisbee)
osload: Issuing reboot for pc167 and then waiting ...
reboot (pc167): Attempting to reboot ...
reboot (pc167): Successful!
reboot: Done. There were 0 failures.
reboot (pc167): child returned 0 status.
osload (pc167): still waiting; it has been 1 minute(s)
osload (pc167): still waiting; it has been 2 minute(s)
```

```
osload (pc167): still waiting; it has been 3 minute(s)
osload (pc167): still waiting; it has been 4 minute(s)
osload: Done! There were 0 failures.
users >
```

I need more disk space on my node

Each node has a partition at the end of the disk that you can use if you wish. However, the naming convention differs between OSes and even between versions of the OS. In Linux the partition is likely to be either `/dev/hda4` (PATA) or `/dev/sda4` (SATA or SCSI), though it is possible that it will be on `/dev/sdb` instead. One slightly more generic way to determine the name is to do `"df /"` to see where the root filesystem is. The extra partition will be on the same disk, but as partition 4 (or 3 if `/dev/sda3` is the last one) rather than 1 or 2. There is no filesystem on this extra partition, so you'll need to create it yourself as described below.

Making a filesystem.

The standard FreeBSD and Linux images also include a script, `/usr/testbed/bin/mkextrafs` to make the remaining space on the root disk (partition 4) available to you. Just do the following on your node:

```
sudo /usr/testbed/bin/mkextrafs /mnt
```

and it will create the filesystem, mount it on `/mnt` and make an entry in `/etc/fstab` so that the filesystem will be mounted on future reboots.

You may need to change the owner of `/mnt`, for example:

```
user=`whoami`
sudo chown $user /mnt
```

****If there is no `/dev/sda4` on your node (i.e. `/dev/sda3` is the last partition), you could simply do the following steps:**

1. Create the file system on `/dev/sda3`

```
sudo mkfs -t ext4 /dev/sda3
```

If you see the following message, enter "y":

/dev/sda3 contains a ext4 file system

last mounted on /mnt/local ...

Proceed anyway? (y,n)

2. Mount the new file system, say, to /mnt

```
sudo mount /dev/sda3 /mnt
```

```
user=`whoami`
```

```
sudo chown $user /mnt
```

3. Check the file system

```
df -h
```

| Filesystem | Size | Used | Avail | Use% | Mounted on |
|------------|------|------|-------|------|------------|
|------------|------|------|-------|------|------------|

| | | | | | |
|-----------|-----|------|-----|-----|---|
| /dev/sda1 | 16G | 2.1G | 13G | 14% | / |
|-----------|-----|------|-----|-----|---|

...

| | | | | | |
|-----------|------|-----|------|----|------|
| /dev/sda3 | 1.8T | 71M | 1.7T | 1% | /mnt |
|-----------|------|-----|------|----|------|

***If that script `/usr/testbed/bin/mkextrafs` **does not exist**, you can also perform the steps by hand. For example, in Ubuntu 14 do the following as root:

1. Delete the existing partition /dev/sda4

```
sudo fdisk /dev/sda
```

type in "d", "4", "w" accordingly.

If WARNING: Re-reading the partition table failed with error 16: Device or resource busy, run:

```
sudo partprobe
```

2. Add a new partition

```
sudo fdisk /dev/sda
```

type in "n", "p", Enter, Enter, "w" accordingly.

If WARNING: Re-reading the partition table failed with error 16: Device or resource busy, run:

```
sudo partprobe
```

3. Create the file system

```
sudo mkfs -t ext4 /dev/sda4
```

4. Mount the new file system, say, to /mnt

```
sudo mount /dev/sda4 /mnt
```

```
user=`whoami`
```

```
sudo chown $user /mnt
```

5. Check the file system

```
df -h
```

| Filesystem | Size | Used | Avail | Use% | Mounted on |
|------------|------|------|-------|------|------------|
| /dev/sda1 | 16G | 2.1G | 13G | 14% | / |
| ... | | | | | |
| /dev/sda4 | 2.0T | 71M | 1.9T | 1% | /mnt |

6. [Optional] Change read/write/exec permission, for example:

```
sudo chmod -R 777 /mnt
```

DONE!



Per-Link Traffic Shaping ("linkdelays")

In order to conserve nodes, it is possible to specify that instead of doing traffic shaping on separate delay nodes (which eats up a node for every two shaped links), it be done on the nodes that are actually generating the traffic.

Under FreeBSD, just like normal delay nodes, end node (sometimes called "per-link") traffic shaping uses IPFW to direct traffic into the proper Dummynet pipe. On each node in a duplex link or LAN, a set of IPFW rules and Dummynet pipes is set up. As traffic enters or leaves your node, IPFW looks at the packet and stuffs it into the proper Dummynet pipe. At the proper time, Dummynet takes the packet and sends it on its way.

Under Linux, end node traffic shaping is performed by the packet scheduler modules, part of the kernel NET3 implementation. Each packet is added to the appropriate scheduler queue tree and shaped as specified in your NS file. Note that Linux traffic shaping currently only supports the drop-tail queueing discipline; gred and red are not available yet.

To specify end node shaping in your NS file, simply set up a normal link or LAN, and then mark it as wanting to use end node traffic shaping. For example:

```
set link0 [$ns duplex-link $nodeA $nodeD 50Mb 0ms DropTail]
set lan0  [$ns make-lan "nodeA nodeB nodeC" 10Mb 100ms]

tb-set-endnodeshaping $link0 1
tb-set-endnodeshaping $lan0 1
```

Please be aware though, that the kernels are different than the standard ones in a couple of ways:

- The kernel runs at a 1000HZ (1024HZ in Linux) clockrate instead of 100HZ. That is, the timer interrupts 1000 (1024) times per second instead of 100. This finer granularity allows the traffic shapers to do a better job of scheduling packets.
- Under FreeBSD, IPFW and Dummynet are compiled into the kernel, which affects the network stack; all incoming and outgoing packets are sent into IPFW to be matched on. Under Linux, packet scheduling exists implicitly, but uses lightweight modules by default.

- The packet timing mechanism in the linkdelay Linux kernel uses a slightly heavier (but more precise) method.
- Flow-based IP forwarding is turned off. This is also known as IP "fast forwarding" in the FreeBSD kernel. Note that regular IP packet forwarding is still enabled.

To use end node traffic shaping globally, without having to specify per link or LAN, use the following in your NS file:

```
tb-use-endnodeshaping    1
```

To specify non-shaped links, but perhaps control the shaping parameters later (increase delay, decrease bandwidth, etc.) after the experiment is swapped in, use the following in your NS file:

```
tb-force-endnodeshaping  1
```

User Do's and Don'ts

If you are a student: DO read the [Student Introduction to NCL Testbed](#).

Preserving our Control Network

- DON'T use control network unless absolutely necessary. This means:
 - DON'T use full node names such as `ping node1.YourExp.YourProj.ncl.sg`
 - DO use short names such as `ping node1`. This ensures that traffic goes over experimental network.
 - DON'T generate traffic to `10.16.x.x` network.
 - DO use addresses of experimental interfaces. These can be from any IPv4 address range, depending on your NS file, but are often from the `172.16.x.x` address range.

Preserving our File System

- DON'T store large files (e.g. uncompressed kernel source) in your home or project directory unless you need them in multiple experiment instances.
 - DO store these files locally on a node, e.g., in `/tmp` folder. If you need more disk space on a Linux or FreeBSD node you can mount more to `/mnt/local` by doing

```
sudo /usr/local/etc/emulab/mkextrafs.pl /mnt
user='whoami'
sudo chown $user /mnt/local
```

Transfer the files to your home directory before you swap out to save them.

- DON'T transfer large (>500 MB) files frequently between your home or project directories and a local directory on your experimental machine.
 - If you need to regularly save and read large files that persist between experiment instances, contact us at support@ncl.sg.
- DON'T perform large (> 500 MB) or frequent (< 10 s) reads/writes on your experimental nodes into your home or project directory
 - DO perform these reads/writes on a local disk (`/tmp` or `/mnt/local` on experimental machines)
- DON'T compile software or kernels in your home directory
 - DO compile them on a local disk (`/tmp` or `/mnt/local` on experimental machines)

Preserving CPU Cycles on users

- DON'T compile large files or run CPU intensive jobs on `users.ncl.sg`.
 - DO allocate experimental nodes, store files locally and compile/run jobs there.

NCL Testbed Commands

The following commands are available from the commandline on `users.ncl.sg`.

Note

Commands should be pre-pended with the path: `/usr/testbed/bin`.

For example, to start an experiment, you would use: `/usr/testbed/bin/startexp`
[options]

Transport Layers

The NCL Testbed XMLRPC server can be accessed via two different transport layers: SSH and SSL.

How to use SSH keys Follow [these directions](#) if you are unfamiliar with using SSH.

How to use SSL You need to request a certificate from the NCL Testbed in order to use the SSL based server.

Operational Commands

startexp: Start an NCL Testbed experiment

```
startexp [-q] [-i [-w]] [-f] [-N] [-E description] [-g gid]
          [-S reason] [-L reason] [-a <time>] [-l <time>]
          -p <pid> -e <eid> <nsfile>
```

Options:

- i Swapin immediately; by default, the experiment is batched.
- w Wait for non-batchmode experiment to preload or swapin.
- f Preload experiment (do not swapin or queue yet).
- q Be less verbose.
- S Experiment cannot be swapped; must provide reason.
- L Experiment cannot be IDLE swapped; must provide reason.
- a Auto swapout NN minutes after experiment is swapped in.
- l Auto swapout NN minutes after experiment goes idle.
- E A concise sentence describing your experiment.
- g The subgroup in which to create the experiment.

-p The project in which to create the experiment.
-e The experiment name (unique, alphanumeric, no blanks).
-N Suppress most email to the user and testbed-ops.
nsfile NS file to parse for experiment

batchexp: Synonym for startexp

This is a legacy command. See command description for startexp.

endexp: Terminate an experiment

```
endexp [-w] [-N] -e pid,eid  
endexp [-w] [-N] pid eid
```

Options:

-w Wait for experiment to finish terminating.
-e Project and Experiment ID.
-N Suppress most email to the user and testbed-ops.

Note

- **Use with caution!** This will tear down your experiment and you will not be able to swap it back in.
 - By default, `endexp` runs in the background, sending you email when the transition has completed. Use the `-w` option to wait in the foreground, returning exit status. Email is still sent.
 - The experiment may be terminated when it is currently swapped in "or" swapped out.
-

delay_config: Change the link shaping characteristics for a link or LAN

```
delay_config [options] -e pid,eid link PARAM#value ...  
delay_config [options] pid eid link PARAM#value ...
```

Options:

-m Modify virtual experiment as well as current state.

-s Select the source of the link to change.
-e Project and Experiment ID to operate on.
link Name of link from your NS file (ie: link1).

Parameters:

BANDWIDTH#NNN N#bandwidth (10-10000000 Kbits per second)
PLR#NNN N#lossrate (0 <# plr < 1)
DELAY#NNN N#delay (one-way delay in milliseconds > 0)
LIMIT#NNN The queue size in bytes or packets
QUEUE-IN-BYTES#N 0 means in packets, 1 means in bytes

RED/GRED Options: (only if link was specified as RED/GRED)

MAXTHRESH#NNN Maximum threshold for the average Q size
THRESH#NNN Minimum threshold for the average Q size
LINTERM#NNN Packet dropping probability
Q_WEIGHT#NNN For calculating the average queue size

modexp: Modify experiment

```
modexp [-r] [-s] [-w] [-N] -e pid,eid nsfile  
modexp [-r] [-s] [-w] [-N] pid eid nsfile
```

Options:

-w Wait for experiment to finish swapping.
-e Project and Experiment ID.
-r Reboot nodes (when experiment is active).
-s Restart event scheduler (when experiment is active).
-N Suppress most email to the user and testbed-ops

Note

- By default, `modexp` runs in the background, sending you email when the transition has completed. Use the `-w` option to wait in the foreground, returning exit status. Email is still sent.

- The experiment can be either swapped in "or" swapped out.
 - If the experiment is swapped out, the new NS file replaces the existing NS file (the virtual topology is updated).
 - If the experiment is swapped in (active), the physical topology is also updated, subject to the `-r` and `-s` options above.

swapexp: Swap experiment in or out

```
swapexp -e pid,eid in|out
```

```
swapexp pid eid in|out
```

Options:

- `-w` Wait for experiment to finish swapping.
- `-e` Project and Experiment ID.
- `-N` Suppress most email to the user and testbed-ops.
- `in` Swap experiment in (must currently be swapped out).
- `out` Swap experiment out (must currently be swapped in)

Note

- By default, `swapexp` runs in the background, sending you email when the transition has completed. Use the `-w` option to wait in the foreground, returning exit status. Email is still sent.

create_image: Create a disk image from a node

```
create_image [options] imageid node
```

Options:

- `-w` Wait for image to be created.
- `-p` Project ID of imageid.
- `imageid` Name of the image.
- `node` Node to create image from (pcXXX).

Example:

The following command creates or re-creates an image for a particular project:

```
create_image -p <proj> <imageid> <node>
```

eventsys_control: Start/Stop/Restart the event system

```
eventsys_control -e pid,eid start|stop|replay
```

```
eventsys_control pid eid start|stop|replay
```

Options:

`-e` Project and Experiment ID.

`stop` Stop the event scheduler.

`start` Start the event stream from time index 0.

`replay` Replay the event stream from time index 0

loghole: Downloads and manages an experiment's log files

This utility downloads log files from certain directories on the experimental nodes (e.g. `/local/logs`) to the NCL Testbed `users` machine. After downloading, it can also be used to produce and manage archives of the log files.

Using this utility to manage an experiment's log files is encouraged because it will transfer the logs in a network-friendly manner and is already integrated with the rest of NCL Testbed. For example, any programs executed using the NCL Testbed event-system will have their standard output/error automatically placed in the `/local/logs` directory. The tool can also be used to preserve multiple trials of an experiment by producing and managing zip archives of the logs.

```
loghole [-hVdqva] [-e pid/eid] [-s server] [-P port] action [...]
```

```
loghole sync [-nPs] [-r remotedir] [-l localdir] [node1 node2 ...]
```

```
loghole validate
```

```
loghole archive [-k (i-delete|space-is-needed)] [-a days] [-c comment]
```

```

    [-d] [archive-name]

loghole change [-k (i-delete|space-is-needed)] [-a days] [-c comment]
    archive-name1 [archive-name2 ...]

loghole list [-O1!Xo] [-m atmost] [-s megabytes]

loghole show [archive-name]

loghole clean [-fne] [node1 node2 ...]

loghole gc [-n] [-m atmost] [-s megabytes]

```

Options:

-h, --help Print the usage message for the loghole utility as a whole or, if an action is given, the usage message for that action.

-V, --version Print out version information and exit.

-d, --debug Output debugging messages.

-q, --quiet Decrease the level of verbosity, this is subtractive, so multiple uses of this option will make the utility quieter and quieter. The default level of verbosity is human-readable, below that is machine-readable, and below that is silent. For example, the default output from the "list" action looks like:

```

[ ] foobar.1.zip    10/15
[!] foobar.0.zip    10/13

```

Using a single **-q** option changes the output to look like:

```

    foobar.1.zip
    foobar.0.zip

```

-e, --experiment#PID/EID Specify the experiment(s) to operate on using the Project ID (or project name) and Experiment ID (or experiment name). If multiple **-e** options are given, the action will apply to all of them. This option overrides the default behavior of inferring the experiment from (Note: this sentence was cut off in the Emulab documentation).

Examples:

To synchronize the log directory for experiment `neptune/myexp` with the log holes on the experimental nodes:

```
[vmars@users ~] loghole -e neptune/myexp sync
```

To archive the newly recovered logs and print out just the name of the new log file:

```
[vmars@users ~] loghole -e neptune/myexp -q archive
```

More information:

To see the detailed documentation of `loghole`, view the man page on `users`:

```
loghole man
```

os_load: Reload disks on selected nodes or all nodes in an experiment

```
os_load [options] node [node ...]
```

```
os_load [options] -e pid,eid
```

Options:

- i Specify image name; otherwise load default image.
- p Specify project for finding image name (-i).
- s Do *not* wait for nodes to finish reloading.
- m Specify internal image id (instead of -i and -p).
- r Do *not* reboot nodes; do that yourself.
- e Reboot all nodes in an experiment.
- node Node to reboot (pcXXX).

portstats: Get portstats from the switches

```
portstats <-p | pid eid> [vname ...] [vname:port ...]
```

Options:

- e Show only error counters.
- a Show all stats.

- z Zero out counts for selected counters after printing.
- q Quiet: don't actually print counts - useful with -z.
- c Print absolute, rather than relative, counts.
- p The machines given are physical, not virtual, node IDs. **No pid and eid should be given with this option.**

Warning

If only the pid and eid are given, this command prints out information about all ports in the experiment. Otherwise, output is limited to the nodes and/or ports given.

Note

- Statistics are reported from the switch's perspective. This means that "In" packets are those sent FROM the node, and "Out" packets are those sent TO the node.
- In the output, packets described as 'NUncast' or 'NUcast' are non-unicast (broadcast or multicast) packets.

node_reboot: Reboot selected nodes or all nodes in an experiment

Use this if you need to powercycle a node.

```
node_reboot [options] node [node ...]
node_reboot [options] -e pid,eid
```

Options:

- w Wait for nodes to come back up.
 - c Reconfigure nodes instead of rebooting.
 - f Power cycle nodes (skip reboot!)
 - e Reboot all nodes in an experiment.
- node Node to reboot (pcXXX).

Note

- You may provide more than one node on the command line.
- Be aware that you may power cycle only nodes in projects that you are member of.

- `node_reboot` does its very best to perform a clean reboot before resorting to cycling the power to the node. This is to prevent the damage that can occur from constant power cycling over a long period of time. For this reason, `node_reboot` may delay a minute or two if it detects that the machine is still responsive to network transmission. In any event, please try to reboot your nodes first (see above).
- You may reboot all the nodes in an experiment by using the `-e` option to specify the project and experiment names. This option is provided as a shorthand method for rebooting large groups of nodes.

Example:

The following command will reboot all of the nodes reserved in the "multicast" experiment in the "testbed" project.

```
node_reboot -e testbed,multicast
```

expwait: Wait for experiment to reach a state

```
expwait [-t timeout] -e pid,eid state
expwait [-t timeout] pid eid state
```

Options:

- `-e` Project and Experiment ID in format `<projectID>/<experimentID>`.
- `-t` Maximum time to wait (in seconds).

Informational Commands

node_list: Print physical mapping of nodes in an experiment

```
node_list [options] -e pid,eid
```

Options:

- `-e` Project and Experiment ID to list.
- `-p` Print physical (NCL database) names (default).
- `-P` Like `-p`, but include node type.

- v Print virtual (experiment assigned) names.
- h Print physical name of host for virtual nodes.
- c Print container VMs and physical nodes.

Note

- This command now queries the XMLRPC interface as it used to do. Users who had been using `script_wrapper.py node_list` to access this function should use `/usr/testbed/bin/node_list` instead.
- The `-c` flag that outputs containerized node names has been modified in two ways.
 - Names are produced without the DNS qualifiers as node names provided by other options of this command are. A node in a VM container named `a` will be reported as `a` not `a.exp.proj` as earlier versions of this feature did.
 - This option now reports `embedded_pnode` containers as well (physical machines). If no containers VMs are present in an experiment, `node_list -c` and `node_list -v` produce identical output.
- The `node_list` command is now available as `node_summary`.

expinfo: Get information about an experiment

```
expinfo [-n] [-m] [-l] [-d] [-a] -e pid,eid
expinfo [-n] [-m] [-l] [-d] [-a] pid eid
```

Options:

- e Project and Experiment ID.
- n Show node information.
- m Show node mapping.
- l Show link information.
- a Show all of the above.

node_avail: Print free node counts

```
node_avail [-p project] [-c class] [-t type]
```

Options:

`-p project` Specify project credentials for node types that are restricted.
`-c class` The node class (Default: pc).
`-t type` The node type.

Example:

The following command will print free nodes on pc850 nodes:

```
$ node_avail -t pc850
```

node_avail_list: Print physical node_ids of available nodes

```
node_avail_list [-p project] [-c class] [-t type] [-n nodes]
```

Options:

`-p project` Specify project credentials for node types that are restricted.
`-c class` The node class (Default: pc).
`-t type` The node type.
`-n pcX,pcY,...,pcZ` A list of physical node_ids.

Example:

The following command will print the physical node_ids for available pc850 nodes:

```
$ node_avail_list -t pc850
```

nscheck: Check and NS file for parser errors

```
nscheck nsfile
```

Option:

`nsfile` Path to NS file you to wish check for parse errors.

NS Commands

In order to use the testbed specific commands, you must include the following line near the top of your NS topology file (before any testbed commands are used):

```
source tb_compat.tcl
```

If you wish to use your file under NS, download [tb_compat.tcl](#) and place it in the same directory as your NS file. When run in this way under NS, the testbed commands will have no effect, but NS will be able to parse your file.

TCL, NS, and node names

In your file, you will be creating nodes with something like the following line:

```
set node1 [$ns node]
```

With this command, the simulator, represented by `$ns` is creating a new node involving many internal data changes and returning a reference to it which is stored in the variable `node1`.

In almost all cases when you need to refer to a node, you will do it as `$node1`, the `$` indicating that you want the value of the variable `node1`, i.e. the reference to the node. Thus you will be issuing commands like:

```
$ns duplex-link $node1 $node2 100Mb 150ms DropTail
tb-set-ip $node1 10.1.0.2
```

Note the instances of `$`.

You will notice that when your experiment is set up, the node names and such will be `node1`, `node2`, `node3`, etc. This happens because the parser detects what variable you are using to store the node reference and uses that as the node name. In the case that you do something like:

```
set node1 [$ns node2]
set A $node1
```

The node will still be called `node1` as that was the first variable to contain the reference.

If you are dealing with many nodes you may store them in an array, using a command similar to the following:

```
for {set i 0} {$i < 4} {incr i} {  
    set nodes($i) [$ns node]  
}
```

In this case, the names of the node will be `nodes-0`, `nodes-1`, `nodes-2`, `nodes-3`. In other words, the "(" character is replaced with "-", and ")" is removed. This slightly different syntax is used to avoid any problems that "(" may cause later in the process. For example, the "(" characters may not appear in DNS entries.

As a final note, everything said above for nodes applies equally to LANs, i.e.:

```
set lan0 [$ns make-lan "$node0 $node1" 10Gb 0ms]  
tb-set-lan-loss $lan0 .02
```

Again, note the instances of `$`.

Links may also be named just like nodes and LANs. The names may then be used to set loss rates or IP addresses. This technique is the only way to set such attributes when there are multiple links between two nodes.

```
set link1 [$ns duplex-link $node0 $node1 100Mb 0ms DropTail]  
tb-set-link-loss $link1 0.05  
tb-set-ip-link $node0 $link1 10.1.0.128
```

Captured NS file parameters

A common convention when writing NS files is to place any parameters in an array named `opt` at the beginning of the file. For example:

```
set opt(CLIENT_COUNT) 5  
set opt(BW) 100mb;      Link bandwidth  
set opt(LAT) 10ms;      Link latency  
  
...
```



```

$ns duplex-link $server $router $opt(BW) $opt(LAT) DropTail

for {set i 0} {$i < $opt(CLIENT_COUNT)} {incr i} {
    set nodes($i) [$ns node]
    ...
}

set serverprog [$server program-agent -command "starter.sh"]

```

Normally, this convention is only used to help organize the parameters. In NCL Testbed, however, the contents of the `opt` array are captured and made available to the emulated environment. For instance, the parameters are added as environment variables to any commands run by program-agents. So in the above example of NS code, the `starter.sh` script will be able to reference parameters by name, like so:

```

#!/bin/sh

echo "Testing with $CLIENT_COUNT clients."
...

```

Note that the contents of the `opt` array are not ordered, so you should not reference other parameters and expect the shell to expand them appropriately:

```

set opt(prefix) "/foo/bar"
set opt(BINDIR) '$prefix/bin'; # BAD

set opt(prefix) "/foo/bar"
set opt(BINDIR) "$opt(prefix)/bin"; # Good

```

Ordering Issues

`tb-` commands have the same status as all other Tcl and NS commands. Thus **the order matters** not only relative to each other but also relative to other commands. One common example of this is that IP commands must be issued *after* the links or LANs are created.

Hardware Commands

tb-set-hardware

```
tb-set-hardware node type [args]
```

```
tb-set-hardware $node3 pc
```

```
tb-set-hardware $node4 shark
```

where:

`node` = The name of the node.

`type` = The type of the node.

Note

- Please see the [Node Status](#) page for a list of available types. `pc` is the default type.
- No current types have any additional arguments.

IP Address Commands

Each node will be assigned an IP address for each interface that is in use. The following commands will allow you to explicitly set those IP addresses. IP addresses will be automatically generated for all nodes for which you do not explicitly set IP addresses.

In most cases, the IP addresses on either side of a link must be in the same subnet. Likewise, all IP addresses on a LAN should be in the same subnet. Generally the same subnet should not be used for more than one link or LAN in a given experiment, nor should one node have multiple interfaces in the same subnet. Automatically generated IP addresses will conform to these requirements. If part of a link or LAN is explicitly specified with the commands below then the remainder will be automatically generated under the same subnet.

IP address assignment is deterministic and tries to fill lower IP's first, starting at 2. Except in the partial specification case (see above), all automatic IP addresses are in the network `172.16.`

tb-set-ip

```
tb-set-ip node ip
```

```
tb-set-ip $node1 142.3.4.5
```

where:

`node` = The node to assign the IP address to

`ip` = The IP address.

Note

- This command should only be used for nodes that have a single link. For nodes with multiple links the following commands should be used. Mixing `tb-set-ip` and any other IP command on the same node will result in an error.

tb-set-ip-link

```
tb-set-ip-link node link ip
```

```
tb-set-ip-link $node0 $link0 142.3.4.6
```

where:

`node` = The node to set the IP for.

`link` = The link to set the IP for.

`ip` = The IP address.

Note

- One way to think of the arguments is a link with the node specifying which side of the link to set the IP for.
- This command cannot be mixed with `tb-set-ip` on the same node.

tb-set-ip-lan

```
tb-set-ip-lan node lan ip
```

```
tb-set-ip-lan $node1 $lan0 142.3.4.6
```

where:

`node` = The node to set the IP for.

`lan` = The lan the IP is on.

`ip` = The IP address.

Note

- One way to think of the arguments is a node with the LAN specifying which port to set the IP address for.
- This command cannot be mixed with `tb-set-ip` on the same node.

tb-set-ip-interface

```
tb-set-ip-interface node dst ip
tb-set-ip-interface $node2 $node1 142.3.4.6
```

where:

`node` = The node to set the IP for.

`dst` = The destination of the link to set the IP for.

`IP` = The IP address.

Note

- This command cannot be mixed on the same node with `tb-set-ip`. (See above)
- In the case of multiple links between the same pair of nodes, there is no way to distinguish which link to set the IP for. This should be fixed soon.
- This command is converted internally to either `tb-set-ip-link` or `tb-set-ip-lan`. It is possible that error messages will report either of those commands instead of `tb-set-ip-interface`.

tb-set-netmask

```
tb-set-netmask lanlink netmask
tb-set-netmask $link0 "255.255.255.248"
```

where:

`lanlink` = The lan or link to set the netmask for.

`netmask` = The netmask in dotted notation.

Note

- This command sets the netmask for a LAN or link. The mask must be big enough to support all of the nodes on the LAN or link!
- You may play with the bottom three octets (0xFFFFXXX) of the mask; attempts to change the upper octets will cause an error.

OS Commands

tb-set-node-os

```
tb-set-node-os node os
```

```
tb-set-node-os $node1 FBSD-STD
```

```
tb-set-node-os $node1 MY_OS
```

where:

`node` = The node to set the OS for.

`os` = The id of the OS for that node.

Note

- The OS may either be one of the standard OS's we provide or a custom OS, created via the web interface.
- If no OS is specified for a node, a default OS is chosen based on the node's type. This is currently 'Ubuntu1404-64-STD' for PCs.
- The currently available standard OS's are listed in the [Testbed Info](#) page -> Standard OS Images.

tb-set-node-rpms

```
tb-set-node-rpms node rpms...
```

```
tb-set-node-rpms $node0 rpm1 rpm2 rpm3
```

Note

- This command sets which RPMs are to be installed on the node when it first boots after being assigned to an experiment.

- Each RPM can be either a path to a file or a URL. Paths must be to files that reside in the `/proj` or `/groups` directory. You are not allowed to place your RPMs in your home directory. `http(s)://` and `ftp://` URLs will be fetched into the experiment's directory, and re-distributed from there.
- See the [Core Guide](#) for more information.

tb-set-node-startcmd

```
tb-set-node-startcmd node startupcmd
```

```
tb-set-node-startcmd $node0 "mystart.sh -a >& /tmp/node0.log"
```

Note

- Specify a script or program to be run when the node is booted.
- See the [Core Guide](#) for more information.

tb-set-node-cmdline

```
tb-set-node-cmdline node cmdline
```

```
tb-set-node-cmdline $node0 {???
```

Note

- Set the cmdline to be passed to the `kernel` when it is booted.
- Currently, this is supported on OSKit kernels only.

tb-set-node-tarfiles

```
tb-set-node-tarfiles node install-dir1 tarfile1 ...
```

The `tb-set-node-tarfiles` command is used to install one or more tar files onto a node's local disk. This command is useful for installing files that are used frequently, but will change very little during the course of your experiments. For example, if your software depends on a third-party library not provided in the standard disk images, you can produce a tarball and have the library ready for use on all the experimental nodes.

Another example would be the data sets for your software. The benefit of installing files using this method is that they will reside on the node's local disk, so your experimental runs will not be disrupted by NFS traffic.

Note

Avoid using this command if the files are changing frequently because the tars are only (re)installed when the nodes boot.

Installing individual tar files or RPMs is a midpoint in the spectrum of getting software onto the experimental nodes. At one extreme, you can read everything over NFS, which works well if the files are changing constantly, but can generate a great deal of strain on the control network and disrupt your experiment. The tar files and RPMs are also read over NFS when the nodes initially boot; however, there won't be any extra NFS traffic while you are running your experiment. Finally, if you need a lot of software installed on a large number of nodes, say greater than 20, it might be best to create a [custom disk image](#). Using a disk image is easier on the control network since it is transferred using multicast, thus greatly reducing the amount of NFS traffic when the experiment is swapped in.

Required Parameters:

- `node` - The node where the files should be installed. Each node has its own tar file list, which may or may not be different from the others.
- One or more `install-dir` and `tarfile` pairs are then listed in the order you wish them to be installed:
 - `install-dir` - An existing directory on the node where the tar file should be unarchived (e.g. `/`, `/usr`, `/usr/local`). The `tar` command will be run as "root" [\[#tb-set-node-tarfiles Note1\]](#), so all of the node's directories will be accessible to you. If the directory does not exist on the image or was not created by the unarchiving of a previous tar file, the installation will fail [\[#tb-set-node-tarfiles Note2\]](#).
 - `tarfile` - An existing tar file located in a project directory (e.g. `/proj` or `/groups`) or an `http`, `https`, or `ftp` URL. In the case of URLs, they are downloaded when the experiment is swapped in and cached in the experiment's directory for future use. In either case, the tar file name is `required` to have one of the following extensions: `.tar`, `.tar.Z`, `.tar.gz`, or `.tgz`. Note that the tar file could have been created anywhere; however, if you want the unarchived files to have

valid NCL Testbed user and group id's, you should create the tar file on `users` or an experimental node.

Example usage:

```
# Overwrite files in /bin and /sbin.
tb-set-node-tarfiles $node0 /bin /proj/foo/mybinmods.tar /sbin
/proj/foo/mysbinmods.tar

# Programmatically generate the list of tarballs.
set tb [list]
# Add a tarball located on a web site.
lappend tb / http://foo.bar/bazzer.tgz
# Add a tarball located in the NCL NFS space.
lappend tb /usr/local /proj/foo/tarfiles/bar.tar.gz
# Use 'eval' to expand the 'tb' list into individual
# arguments to the tb-set-node-tarfiles command.
eval tb-set-node-tarfiles $node1 $tb
```

See also:

- `tb-set-node-rpms`
- [Custom disk images](#)

Note

- Because the files are installed as root, care must be taken to protect the tar file so it cannot be replaced with a trojan that allowed less privileged users to become root.
- Currently, you can only tell how/why an installation failed by examining the node's console log on bootup.

Link Loss Commands

This is the NS syntax for creating a link:

```
$ns duplex-link $node1 $node2 100Mb 150ms DropTail
```


Note

This does not allow for specifying link loss rates. NCL Testbed does, however, support link loss. The following commands can be used to specify link loss rates.

tb-set-link-loss

```
tb-set-link-loss src dst loss
tb-set-link-loss link loss

tb-set-link-loss $node1 $node2 0.05
tb-set-link-loss $link1 0.02
```

where:

`src, dst` = Two nodes to describe the link.
`link` = The link to set the rate for.
`loss` = The loss rate (between 0 and 1).

Note

- There are two syntaxes available. The first specifies a link by a source/destination pair. The second explicitly specifies the link.
- The source/destination pair is incapable of describing an individual link in the case of multiple links between two nodes. Use the second syntax for this case.

tb-set-lan-loss

```
tb-set-lan-loss lan loss

tb-set-lan-loss $lan1 0.3
```

Where:

`lan` = The lan to set the loss rate for.
`loss` = The loss rate (between 0 and 1).

Note

This command sets the loss rate for the entire LAN.

tb-set-node-lan-delay

```
tb-set-node-lan-delay node lan delay
```

```
tb-set-node-lan-delay $node0 $lan0 40ms
```

Where:

`node` = The node we are modifying the delay for.

`lan` = Which LAN the node is in that we are affecting.

`delay` = The new node to switch delay (see below).

Note

- This command changes the delay between the node and the switch of the LAN. This is only half of the trip a packet must take. The packet will also traverse the switch to the destination node, possibly incurring additional latency from any delay parameters there.
- If this command is not used to overwrite the delay, then the delay for a given node to switch link is taken as one half of the delay passed to `make-lan`. Thus in a LAN where no `tb-set-node-delay` calls are made, the node-to-node latency will be the latency passed to `make-lan`.
- The behavior of this command is not defined when used with nodes that are in the same LAN multiple times.
- Delays of less than 2ms (per trip) are too small to be accurately modeled at this time, and will be silently ignored. As a convenience, a delay of 0ms can be used to indicate that you do not want added delay; the two interfaces will be "directly" connected to each other.

tb-set-node-lan-bandwidth

```
tb-set-node-lan-bandwidth node lan bandwidth
```

```
tb-set-node-lan-bandwidth $node0 $lan0 20Mb
```

Where:

`node` = The node we are modifying the bandwidth for.

`lan` = Which LAN the node is in that we are affecting.

`bandwidth` = The new node to switch bandwidth (see below).

Note

- This command changes the bandwidth between the node and the switch of the LAN. This is only half of the trip a packet must take. The packet will also traverse the switch to the destination node which may have a lower bandwidth.
- If this command is not used to overwrite the bandwidth, then the bandwidth for a given node to switch link is taken directly from the bandwidth passed to `make-lan`.
- The behavior of this command is not defined when used with nodes that are in the same LAN multiple times.

tb-set-node-lan-loss

```
tb-set-node-lan-loss node lan loss
```

```
tb-set-node-lan-loss $node0 $lan0 0.05
```

Where:

`node` = The node we are modifying the loss for.

`lan` = Which LAN the node is in that we are affecting.

`loss` = The new node to switch loss (see below).

Note

- This command changes the loss probability between the node and the switch of the LAN. This is only half of the trip a packet must take. The packet will also traverse the switch to the destination node which may also have a loss chance. Thus for packet going to switch with loss chance A and then going on the destination with loss chance B , the node-to-node loss chance is $(1 - (1-A)(1-B))$.
- If this command is not used to overwrite the loss, then the loss for a given node to switch link is taken from the loss rate passed to the `make-lan` command. If a loss rate of L is passed to `make-lan` then the node to switch loss rate for each node is set to $(1 - \sqrt{1-L})$. Because each packet will have two such chances to be lost, the node-to-loss rate comes out as the desired L .
- The behavior of this command is not defined when used with nodes that are in the same LAN multiple times.

tb-set-node-lan-params

```
tb-set-node-lan-params node lan delay bandwidth loss
```

```
tb-set-node-lan-params $node0 $lan0 40ms 20Mb 0.05
```

Where:

`node` = The node we are modifying the loss for.

`lan` = Which LAN the node is in that we are affecting.

`delay` = The new node to switch delay.

`bandwidth` = The new node to switch bandwidth.

`loss` = The new node to switch loss.

Note

This command is exactly equivalent to calling each of the above three commands appropriately. See above for more information.

tb-set-link-simplex-params

```
tb-set-link-simplex-params link src delay bw loss
```

```
tb-set-link-simplex-params $link1 $srcnode 100ms 50Mb 0.2
```

Where:

`link` = The link we are modifying.

`src` = The source, defining which direction we are modifying.

`delay` = The source to destination delay.

`bw` = The source to destination bandwidth.

`loss` = The source to destination loss.

Note

- This command modifies the delay characteristics of a link in a single direction. The other direction is unchanged.
- This command only applies to links. Use `tb-set-lan-simplex-params` below for LANs.

tb-set-lan-simplex-params

```
tb-set-lan-simplex-params lan node todelay tobw toloss fromdelay frombw  
fromloss
```

```
tb-set-lan-simplex-params $lan1 $node1 100ms 10Mb 0.1 5ms 100Mb 0
```

Where:

lan = The lan we are modifying.

node = The member of the lan we are modifying.

tdelay = Node to lan delay.

tobw = Node to lan bandwidth.

toloss = Node to lan loss.

fromdelay = Lan to node delay.

frombw = Lan to node bandwidth.

fromloss = Lan to node loss.

Note

This command is exactly like `tb-set-node-lan-params` except that it allows the characteristics in each direction to be chosen separately. See all the notes for `tb-set-node-lan-params`.

tb-set-endnodeshaping

```
tb-set-endnodeshaping link-or-lan enable
```

```
tb-set-endnodeshaping $link1 1
```

```
tb-set-endnodeshaping $lan1 1
```

Where:

link-or-lan = The link or LAN we are modifying.

enable = Set to 1 to enable, 0 to disable.

Note

- This command specifies whether end node shaping is used on the specified link or LAN (instead of a delay node).
- Disabled by default for all links and LANs.

- Only available when running the standard NCL Testbed FreeBSD or Linux kernels.
- See [End Node Traffic Shaping and Multiplexed Links](#) for more details.

tb-set-noshaping

```
tb-set-noshaping link-or-lan enable
```

```
tb-set-noshaping $link1 1
```

```
tb-set-noshaping $lan1 1
```

Where:

`link-or-lan` = The link or LAN we are modifying.

`enable` = Set to 1 to disable bandwidth shaping, 0 to enable.

Note

- This command specifies whether link bandwidth shaping should be enforced on the specified link or LAN. When enabled, bandwidth limits indicated for a link or LAN will not be enforced.
- Disabled by default for all links and LANs. That is, link bandwidth shaping *is* enforced on all links and LANs by default.
- If the delay and loss values for a `tb-set-noshaping` link are zero (the default), then no delay node or end-node delay pipe will be associated with the link or LAN.
- **This command is a hack.** The primary purpose for this command is to subvert the topology mapper (`assign`). `Assign` always observes the physical bandwidth constraints of the testbed. By using `tb-set-noshaping`, you can convince `assign` that links are low-bandwidth and thus get your topology mapped, but then not actually have the links shaped.

tb-use-endnodeshaping

```
tb-use-endnodeshaping enable
```

```
tb-use-endnodeshaping 1
```

Where:

`enable` = Set to 1 to enable end-node traffic shaping on all links and LANs.

Note

- This command allows you to use end-node traffic shaping globally, without having to specify per link or LAN with `tb-set-endnodeshaping`.
- See [End Node Traffic Shaping and Multiplexed Links](#) for more details.

tb-force-endnodeshaping

```
tb-force-endnodeshaping enable
```

```
tb-force-endnodeshaping 1
```

Where:

`enable` = Set to 1 to force end-node traffic shaping on all links and LANs.

Note

- This command allows you to specify non-shaped links and LANs at creation time, but still control the shaping parameters later (e.g., increase delay, decrease bandwidth) after the experiment is swapped in.
- This command forces allocation of end-node shaping infrastructure for all links. There is no equivalent to force delay node allocation.
- See [End Node Traffic Shaping and Multiplexed Links](#) for more details.

tb-set-multiplexed

```
tb-set-multiplexed link allow
```

```
tb-set-multiplexed $link1 1
```

Where:

`link` = The link we are modifying.

`'allow'` = Set to 1 to allow multiplexing of the link, 0 to disallow.

Note

- This command allows a link to be multiplexed over a physical link along with other links.
- Disabled by default for all links.
- Only available when running the standard NCL FreeBSD (not Linux) and only for links (not LANs).
- See [End Node Traffic Shaping and Multiplexed Links](#) for more details.

tb-set-vlink-emulation

```
tb-set-vlink-emulation style
```

```
tb-set-vlink-emulation $link1 vlan
```

Where:

style = One of "vlan" or "veth-ne"

Note

It seems to be necessary to set the virtual link emulation style to vlan for multiplexed links to work under linux.

Misc. Commands

tb-fix-node

```
tb-fix-node vnode pnode
```

```
tb-fix-node $node0 pc42
```

Where:

vnode = The node we are fixing.

pnode = The physical node we want used.

Note

- This command forces the virtual node to be mapped to the specified physical node. Swap in will fail if this cannot be done.

- Do not use this command on nodes that are a virtual type.

tb-fix-interface

```
tb-fix-interface vnode vlink iface
```

```
tb-fix-interface $node0 $link0 "eth0"
```

Where:

`vnode` = The node we are fixing.

`vlink` = The link connecting to that node that we want to set.

`iface` = The NCL Testbed name for the interface that is to be used.

Note

- The interface names used are the ones in the NCL Testbed database - we can make no guarantee that the OS image that boots on the node assigns the same name.
- Different types of nodes have different sets of interfaces, so this command is most useful if you are also using `tb-fix-node` and/or `tb-set-hardware` on the `vnode`.

tb-set-uselatestwadata

```
tb-set-uselatestwadata 0
```

```
tb-set-uselatestwadata 1
```

Note

- This command indicates which widearea data to use when mapping widearea nodes to links. The default is 0, which says to use the aged data. Setting it to 1 says to use the most recent data.

tb-set-wasolver-weights

```
tb-set-wasolver-weights delay bw plr
```

```
tb-set-wasolver-weights 1 10 500
```

Where:

`delay` = The weight to give delay when solving.
`bw` = The weight to give bandwidth when solving.
`plr` = The weight to give lossrate when solving.

Note

- This command sets the relative weights to use when assigning widearea nodes to links. Specifying a zero says to ignore that particular metric when doing the assignment. Setting all three to zero results in an essentially random selection.

tb-set-node-failure-action

```
tb-set-node-failure-action node action

tb-set-node-failure-action $nodeA "fatal"
tb-set-node-failure-action $nodeB "nonfatal"
```

Where:

`node` = The node name.
`action` = One of "fatal" or "nonfatal".

Note

This command sets the failure mode for a node. When an experiment is swapped in, the default action is to abort the swapin if any nodes fail to come up normally. This is the "fatal" mode. You may also set a node to "nonfatal" which will cause node bootup failures to be reported, but otherwise ignored during swapin. Note that this can result in your experiment not working properly if a dependent node fails, but typically you can arrange your software to deal with this.

Understanding Swapping (Node Use Policies)

What are NCL Testbed's use policies?

As a courtesy to other experimenters, we ask that experiments be swapped out or terminated when they are no longer in active use. There are a limited number of nodes available, and node reservations are exclusive, so it is important to free nodes that will be idle so that others may use them. In summary, our policy is that experiments should be swapped out when they are not in use. We encourage you to do that yourself. In general, if experiments are idle for several hours, the system will automatically swap them out, or send you mail about it, and/or an operator may manually swap them out. The actual grace period will differ depending on the size of the experiment, the current demand for resources, and other factors (such as whether you've been a good NCL Testbed citizen in the past!). If you mark your experiment "non-idle-swappable" at creation time or before swapin, and testbed-ops approves your justification, we will make every effort to contact you before swapping it, since local node state could be lost on swapout. Please see full details below.

What is "active use"?

A node or experiment that is being actively used will be doing something related to your experiment. In almost all cases, someone will either be logged into it using it interactively, or some program will be running, sending and receiving packets, and performing the operations necessary to carry out the experiment.

When is an experiment considered idle?

Your experiment will be considered idle if it has no measurable activity for a significant period of time (a few hours; the exact time is typically set at swapin time). We detect the following types of activity:

- Any network activity on the experimental network
- Substantial activity on the control network
- TTY/console activity on nodes
- High CPU activity on nodes
- Certain external events, such as rebooting a node with `node_reboot`

If your experiment's activity falls outside these measured types of activity, or it seems that NCL Testbed is not assessing your idle time correctly, please be sure to let us know when you create your experiment, or you may be swapped out unexpectedly.

"It is considered abuse to generate artificial activity in order to prevent your experiment from being marked idle. Abusers' access to NCL Testbed will be revoked, and their actions will be reported to their project leader. Please do not do this. If you think you need special assistance for a deadline, demo or other reason, please [contact us](#)."

What is "swapping"?

Swapping is the process of instantiating your experiment, i.e., allocating nodes, configuring links, etc. It also refers to the reverse process, in which nodes are released. These processes are called "swapping in" and "swapping out" respectively.

What is an "Idle-Swap"?

An "Idle-Swap" is when NCL Testbed or its operators swap out your experiment because it was idle for too long. There are two ways that your experiment may be idle-swapped: automatic and manual.

The most common is automatic, which happens when Idle-Swap is enabled for your experiment and the experiment has been continuously idle for the idle-swap time that was set at creation/swapin time (usually a few hours). NCL Testbed will then automatically swap it out.

The other way to get idle-swapped is manually, by a NCL Testbed operator. This typically happens when there is very high resource demand and the experiment has been idle a substantial time, usually a few hours. In this case we will typically make every effort to contact you, since it may cause you to lose data stored on the nodes.

"Note that operators (and you) may swap your excessively idle experiment whether or not it is marked idle-swappable."

When you create your experiment, you may uncheck the "Idle-Swap" box, disabling the automatic idle-swapping of your experiment. If you do so, you must specify the reason, which will be reviewed by testbed-ops. If your reason is judged unacceptable or insufficient, we will explain why, and your experiment will be marked idle-swappable. Valid reasons might be things such as:

- "Your idle-detection system fails to detect my experimental activity."
- "I have node-local state that is impractical to copy off in a timely or reliable manner, because"
- "My experiment takes a huge number of nodes, I have several runs to make with intervening think time, and if someone grabs some of these nodes if I'm swapped while thinking, I'll miss my deadline 2 days from now."

If an experiment is non-idle-swappable, our system will not automatically swap it out, and testbed administrators will attempt to contact you in the event a swapout becomes necessary. However, we expect you to be responsible for managing your experiment in a responsible way, a way that uses NCL Testbed's hardware resources efficiently.

When you create your experiment, you may decrease the idle-swap time from the displayed default, but you may not raise it. If lowering it is compatible with your planned use, doing so helps you be a good NCL Testbed citizen. If you want it raised, for example for reasons similar to those given above, send mail to support@ncl.sg.

You may edit the swap settings (Idle-Swap, Max-Duration, and corresponding reasons and timeouts) using the "Modify Experiment" button for your experiment.

How long is too long for a node to be idle?

Ideally, an experiment should be used nearly continuously from start to finish of the experiment, then swapped out or terminated. However, this isn't always possible. In general, if your experiment is idle for 2 hours or more, it should be swapped out. This is especially true at night (in SG timezones) and on weekends. Many experimenters take advantage of lower demand during evenings and weekends to run their large-scale (50-150 node) tests. If your experiment uses 10 nodes or more, it is even more important to release your nodes as soon as possible. Swapin and swapout only take a few minutes (typically 15 for swapin, and less than 1 for swapout), so you won't lose much time by doing it.

Sometimes an experiment will run long enough that you cannot be online to terminate it, for example, if the experiment completes in the middle of the night. We provide three mechanisms to assist you in terminating your experiment and releasing nodes in a timely manner. The first is the Idle Swap, explained above, the second is [Scheduled Termination](#), and the third is the "Max Duration" option, [explained below](#).

What is "node state"?

Some experiments have state that is stored exclusively on the nodes themselves, on their local hard drives. This is state that is not in your NS file or files or disk images that it references, and therefore is not preserved in our database across swapin/swapout. This is state you add to your machines "by hand" after NCL Testbed sets up your experiment, like files you add or modify on filesystems local to test nodes. Local node state does not include any data you store in `/users`, `/proj`, or `/groups`, since those are saved on a fileserver, and not on the local nodes.

Most experiments don't have any local node state, and can be swapped out and in without losing any info. This is highly recommended, since it is more courteous to other experimenters. It allows you or NCL Testbed to easily free up your nodes at any time without losing any of your work. "Please make your experiments adhere to this guideline whenever possible."

An experiment that needs local state that inherently cannot be saved (for some reason) or that you will not be able to copy off before your experiment hits the "idle-swap time," should not be marked "idle-swap" when you create it. In the "Create Experiment" form you must explain the reason. If you must have node state, you can save it before you swap out by copying it by hand (e.g., into a tar or RPM file), or creating a disk image of the node in question, and later reloading it to a new node after you swap in again. Disk images in effect create a "custom OS" that may be loaded automatically based on your NS file. More information about disk images can be found on our [Disk Image page](#).

I just received an email asking me to swap or terminate my experiment.

NCL Testbed has a system for detecting node use, to help achieve more efficient and fair use of NCL Testbed's limited resources. This system sends email messages to experiment leaders whose experiments have been idle for several hours. If you get a message like this, your experiment has been inactive for too long and you should free up its nodes. If the experiment continues to be idle, more reminders may be sent, and soon your project leader will be one of the recipients. After you have been notified, your experiment may be swapped at any time, depending on current demand for nodes, and other factors.

If you feel you received the message in error, please respond to Testbed Operations (testbed-ops@ncl.sg) as soon as possible, describing how you have used your node in the last few hours. There are some types of activity that are difficult to accurately detect, so we'd like to know how we can improve our activity detection system. "Above all, do not ignore these

messages." If you get several reminders and don't respond, your experiment will be swapped out, potentially causing loss of some of your work (see "node state" above). If there is a reason you need to keep your experiment running, tell us so we don't inadvertently cause problems for you.

Someone swapped my experiment!

As described above, the system automatically swaps out your experiment after it reaches its idle time limit, or sometimes an NCL Testbed operator does it earlier when resources are in especially high demand. In the latter case, we will typically try to contact you by email before we swap it out. However, especially if the experiment has been idle for several hours, we may swap it out for you without waiting very long to hear from you. Because of this, it is critical that you keep in close contact with us about an experiment that we may perceive as idle if you want to avoid any loss of your work.

What is "Max duration"?

Each experiment may have a Maximum Duration, where an experimenter specifies the maximum amount of time that the experiment should stay swapped in. When that time is exceeded, the experiment is unconditionally swapped out. The timer is reset every time the experiment swaps in. A reminder message is sent about an hour before the experiment is swapped. This swapout happens regardless of any activity on the nodes, and can be averted by using the "Modify Experiment" button on the Experiments page to turn off the Maximum Duration feature or to lengthen the duration.

This feature allows users to schedule experiment swapouts, helping them to release nodes in a timely manner. For instance, if you plan to use your experiment throughout an 8 hour work day, you can schedule a swapout for 8 hours after it is swapped in. That way, if you forget to swap out before leaving for the day, it will automatically free up the nodes for other users, without leaving the nodes idle for several hours before being idle-swapped, and will work even if you leave your test programs running, making the experiment look non-idle. For automated experiments, it lets you schedule a swapout for slightly after the maximum amount of time your experiment should last. It can also help catch "runaway" experiments (typically batch).

"Max duration" has a similar effect as [scheduled termination/swapout](#), which is specified in the NS file. The differences are that the former lets you adjust the duration while the experiment is running, you get a warning email, and you're always swapped, never terminated. (It's also implemented differently, with a 5 minute scheduling granularity.)

Accessing testbed nodes using SSH

Each node on the testbed is reachable via [SSH](#). One main difference between NCL and Emulab (and other public cloud services like Amazon EC2) is that NCL nodes are not accessible directly from the internet. In order to log into your nodes, you must first log into "users.ncl.sg" using your NCL username (not your email address) and password (or your SSH public key). From `users` you can log into your nodes. To save on connections, you might want to look into using [GNU screen](#) on `users`. Also refer to the [Tips and Tricks](#) section below for ways to make accessing NCL easier.

Uploading files to NCL

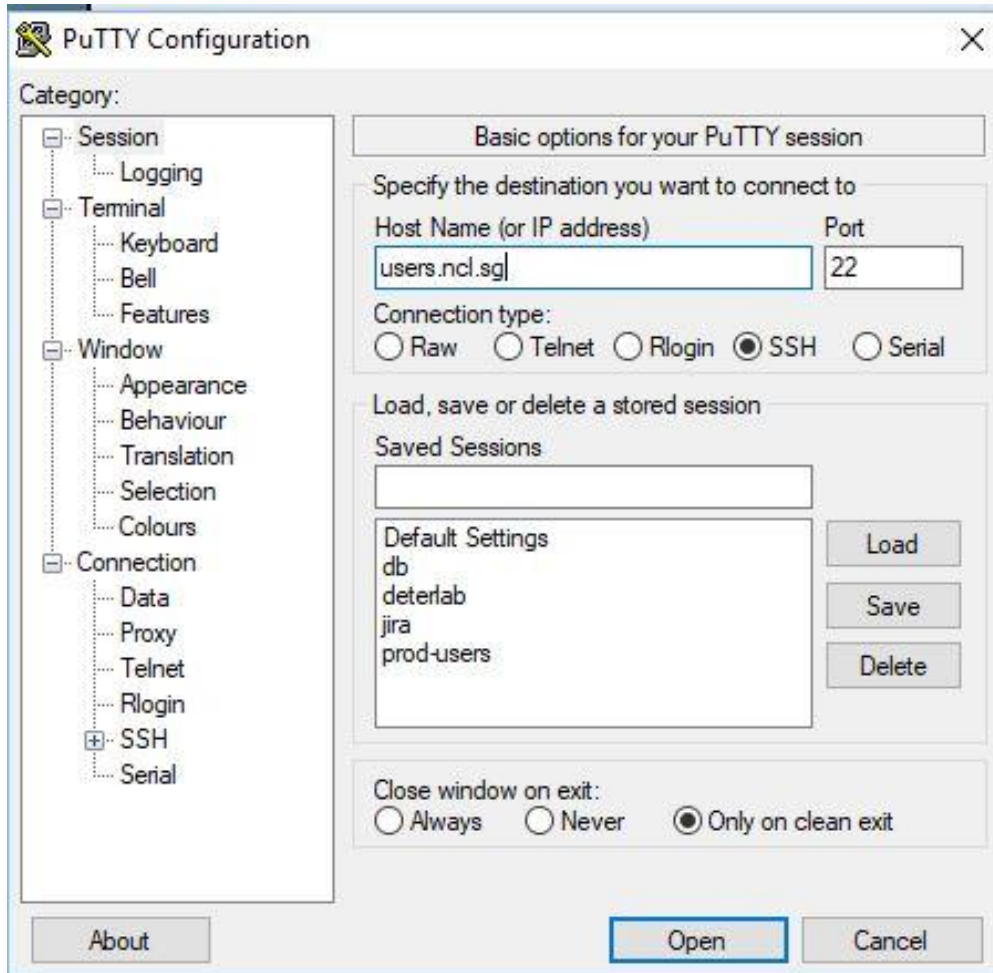
You can upload files to users.ncl.sg via [SCP](#) or [SFTP](#). Files in your home directory and in your project directory will be made available to you on all of your testbed nodes via [NFS](#).

NOTE: There is a disk quota limit on how many files you can store on your home or project directory. The current limit is 25GB. If you have reached that limit, you need to clear files in your home directory or project directory, or move files to your nodes' local storage (see [I need more disk space on my node](#)).

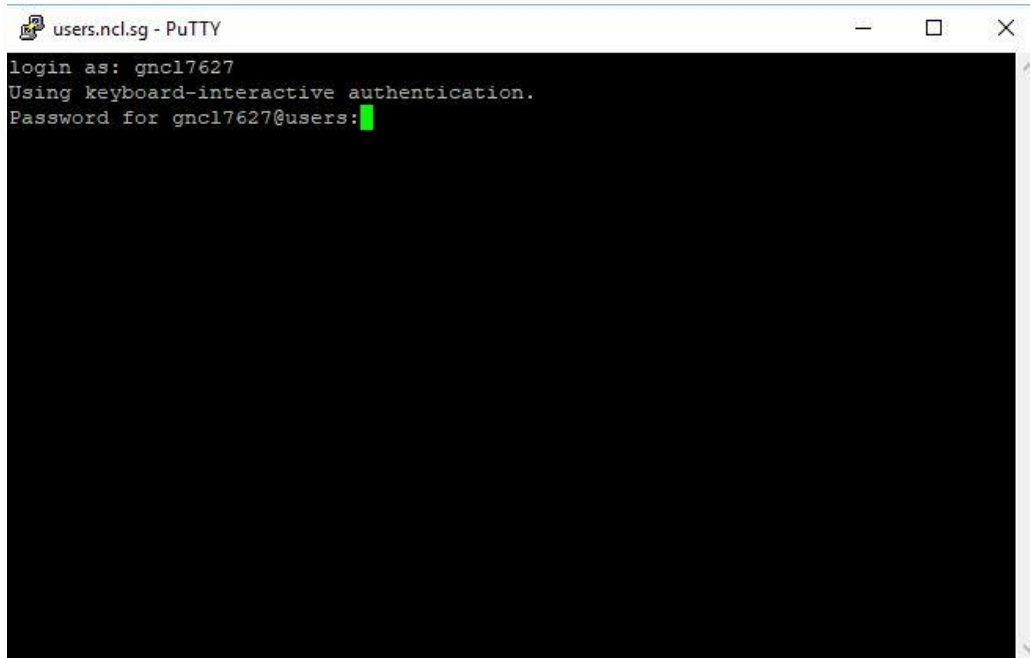
An Example Session with Windows and Putty

[Putty](#) is a free, lightweight SSH client for Windows. Here is an example session in which I connect to my experimental node "nodeA" in my experiment "basic-experiment" in the project "ncltest01".

First we connect to "users.ncl.sg":

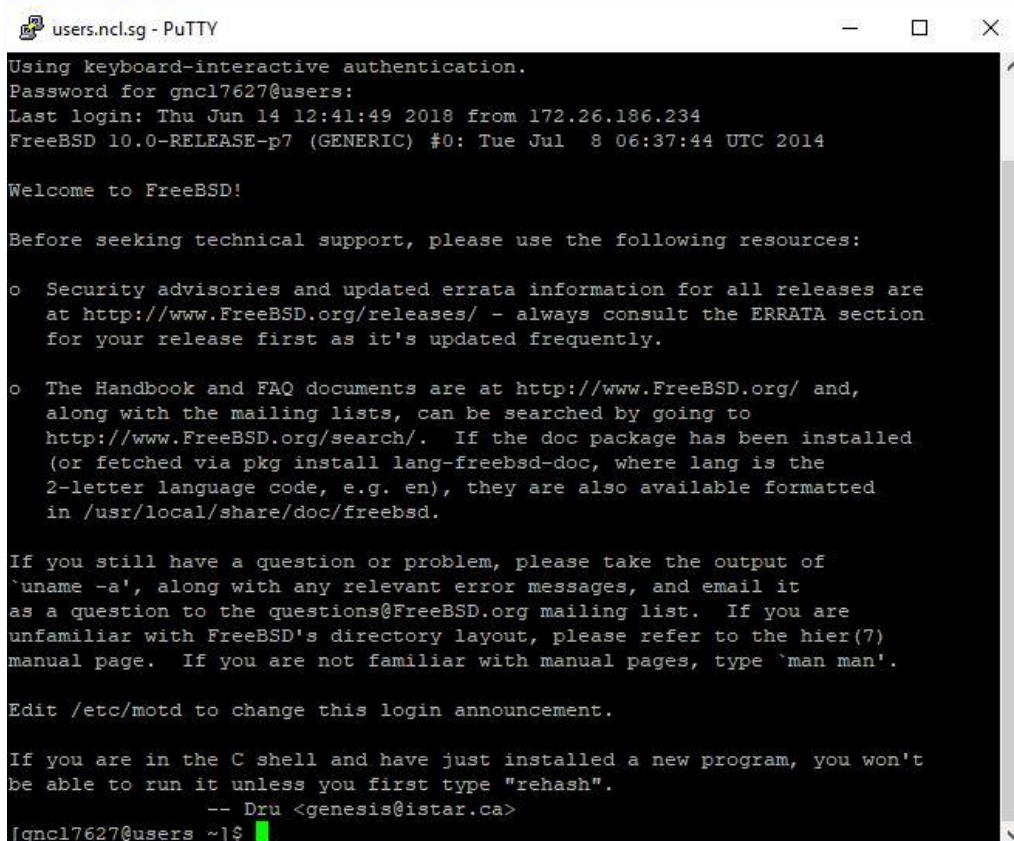


We then enter in our username and password (the same password as the NCL Testbed web interface). Trying to use your email address will "not" work:



```
users.ncl.sg - PuTTY
login as: gnc17627
Using keyboard-interactive authentication.
Password for gnc17627@users:█
```

Now we have successfully logged into users.ncl.sg:



```
users.ncl.sg - PuTTY
Using keyboard-interactive authentication.
Password for gnc17627@users:
Last login: Thu Jun 14 12:41:49 2018 from 172.26.186.234
FreeBSD 10.0-RELEASE-p7 (GENERIC) #0: Tue Jul 8 06:37:44 UTC 2014

Welcome to FreeBSD!

Before seeking technical support, please use the following resources:

o Security advisories and updated errata information for all releases are
  at http://www.FreeBSD.org/releases/ - always consult the ERRATA section
  for your release first as it's updated frequently.

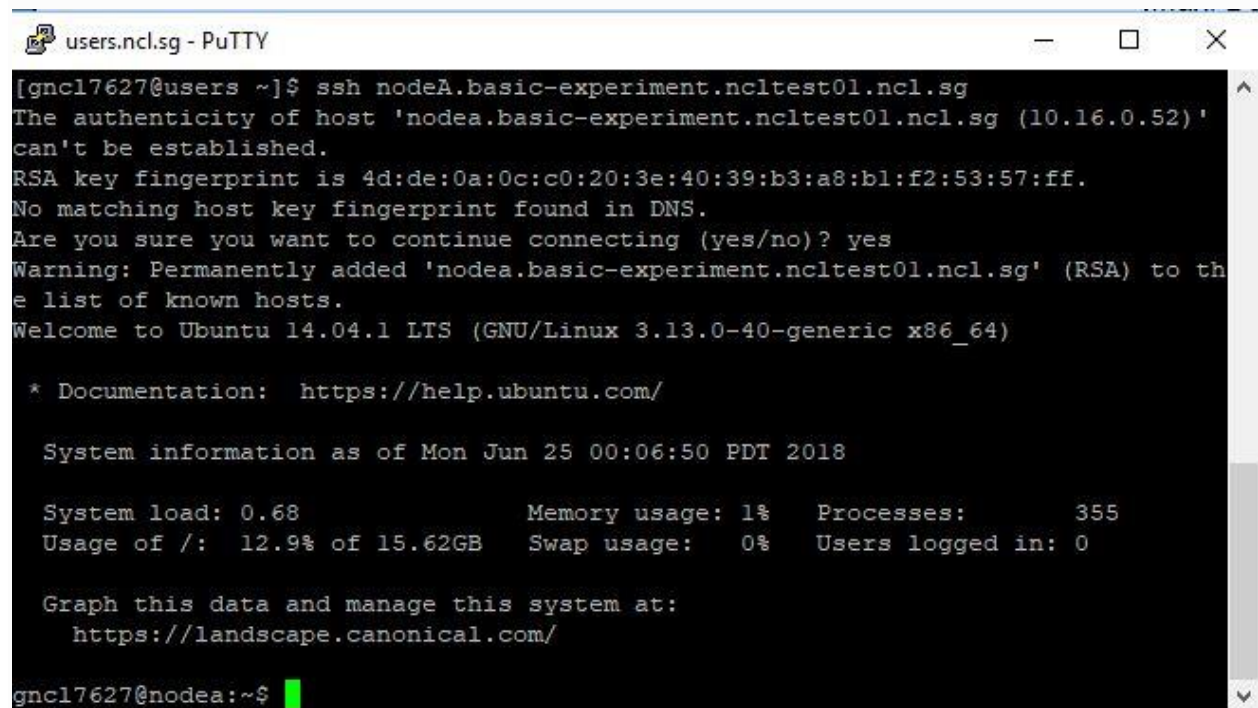
o The Handbook and FAQ documents are at http://www.FreeBSD.org/ and,
  along with the mailing lists, can be searched by going to
  http://www.FreeBSD.org/search/. If the doc package has been installed
  (or fetched via pkg install lang-freebsd-doc, where lang is the
  2-letter language code, e.g. en), they are also available formatted
  in /usr/local/share/doc/freebsd.

If you still have a question or problem, please take the output of
`uname -a`, along with any relevant error messages, and email it
as a question to the questions@FreeBSD.org mailing list. If you are
unfamiliar with FreeBSD's directory layout, please refer to the hier(7)
manual page. If you are not familiar with manual pages, type `man man`.

Edit /etc/motd to change this login announcement.

If you are in the C shell and have just installed a new program, you won't
be able to run it unless you first type "rehash".
-- Dru <genesis@istar.ca>
[gnc17627@users ~]$█
```

From users, we now ssh into our experimental node, "nodeA.basic-experiment.ncltest01.ncl.sg":



```
users.ncl.sg - PuTTY
[gncl7627@users ~]$ ssh nodeA.basic-experiment.ncltest01.ncl.sg
The authenticity of host 'nodea.basic-experiment.ncltest01.ncl.sg (10.16.0.52)'
can't be established.
RSA key fingerprint is 4d:de:0a:0c:c0:20:3e:40:39:b3:a8:b1:f2:53:57:ff.
No matching host key fingerprint found in DNS.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'nodea.basic-experiment.ncltest01.ncl.sg' (RSA) to th
e list of known hosts.
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-40-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Jun 25 00:06:50 PDT 2018

System load: 0.68           Memory usage: 1%    Processes:      355
Usage of /:  12.9% of 15.62GB Swap usage:   0%    Users logged in: 0

Graph this data and manage this system at:
  https://landscape.canonical.com/

gncl7627@nodea:~$
```

Use MobaXterm on Windows

[MobaXterm](#) is another easy-to-use SSH terminal tool on Windows and is actually our recommended SSH client for Windows. For using MobaXterm to access the NCL nodes via SSH, please refer to this [tutorial](#).

Tips and Tricks

Listing your nodes from the command line

When logged into `users.ncl.sg`, the `node_list` command will list the names of all your nodes.

You can log into your nodes using either the `pcXXX` name or the full experimental name.

```
[gncl7627@users ~]$ node_list -e ncltest01,basic-experiment
pc21a  pc22f  pc21f  pc4f  pc24a
```

SSH port forwarding

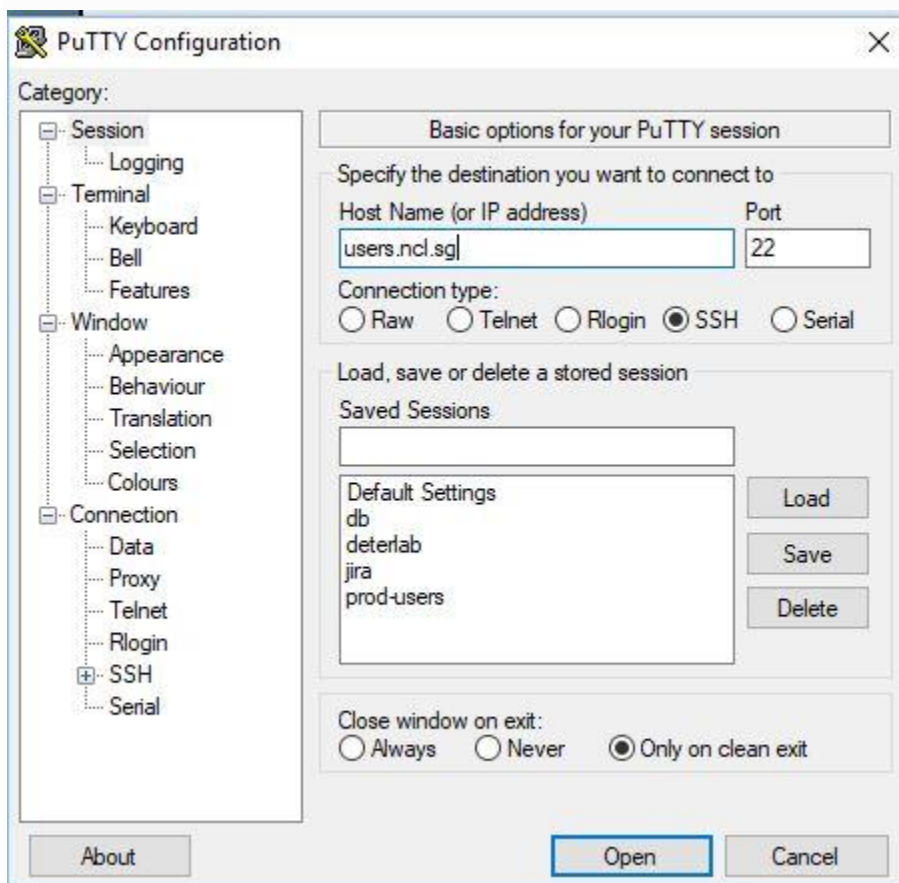
If you are, for example, running an internal web server on one of your NCL nodes, you can access it via SSH through users. For example to redirect port 80 on pcXXX to your local machine on port 8080 you would do:

```
ssh -L 8080:pcXXX:80 username@users.ncl.sg
```

Once logged in, you should be able to access the web server on your NCL node by going to <http://localhost:8080>. For more information on port forwarding with SSH, please refer to the SSH man page.

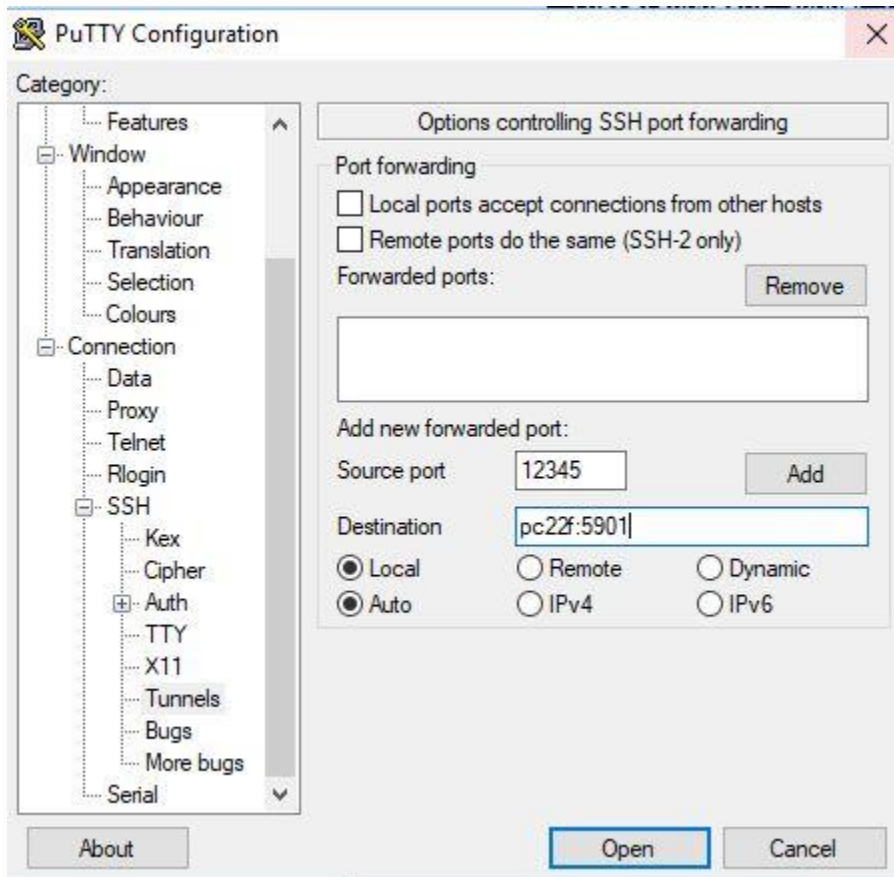
SSH port forwarding with Putty

To use putty for port forwarding, configure putty to open a connection to `users.ncl.sg`

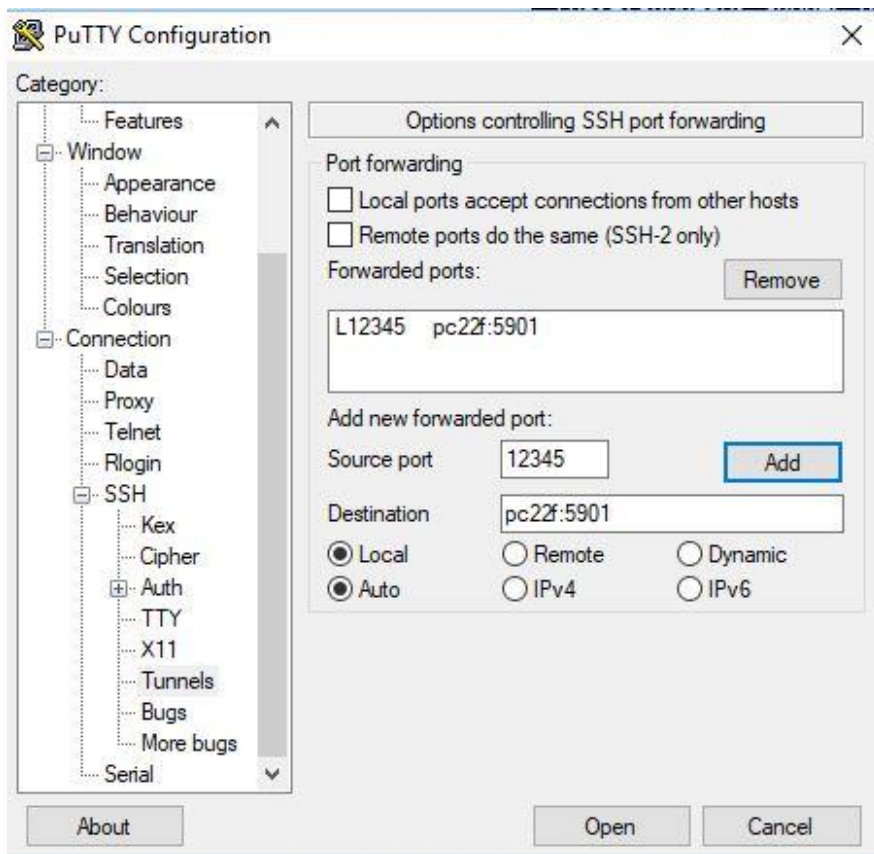


Before you make that connection, set up the tunneling parameters. This example forwards local port 12345 to a remote VNC server (port 5901) on a testbed node. Select the `Tunnels` menu

from under the `SSH` choice in the `Connection` menu on the left hand side. Add a forwarded port using the `Local` type, a local port number (12345 in the image) and the NCL hostname and port in the `Destination` field. In the example we are forwarding the connection to port 5901 (the VNC server) on `pc22f`. (Make sure that your VNC server is up and running on the port 5901)



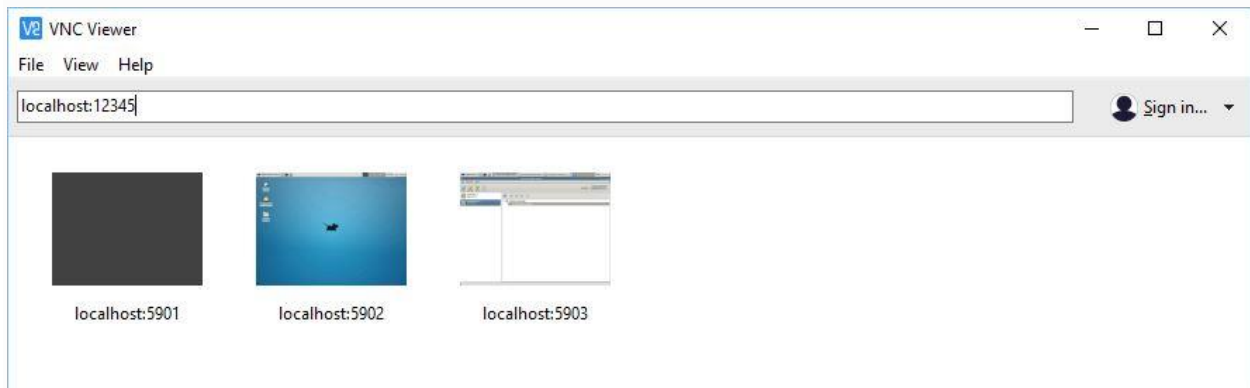
Be sure to press `Add` to add the port. The putty window will look like this:



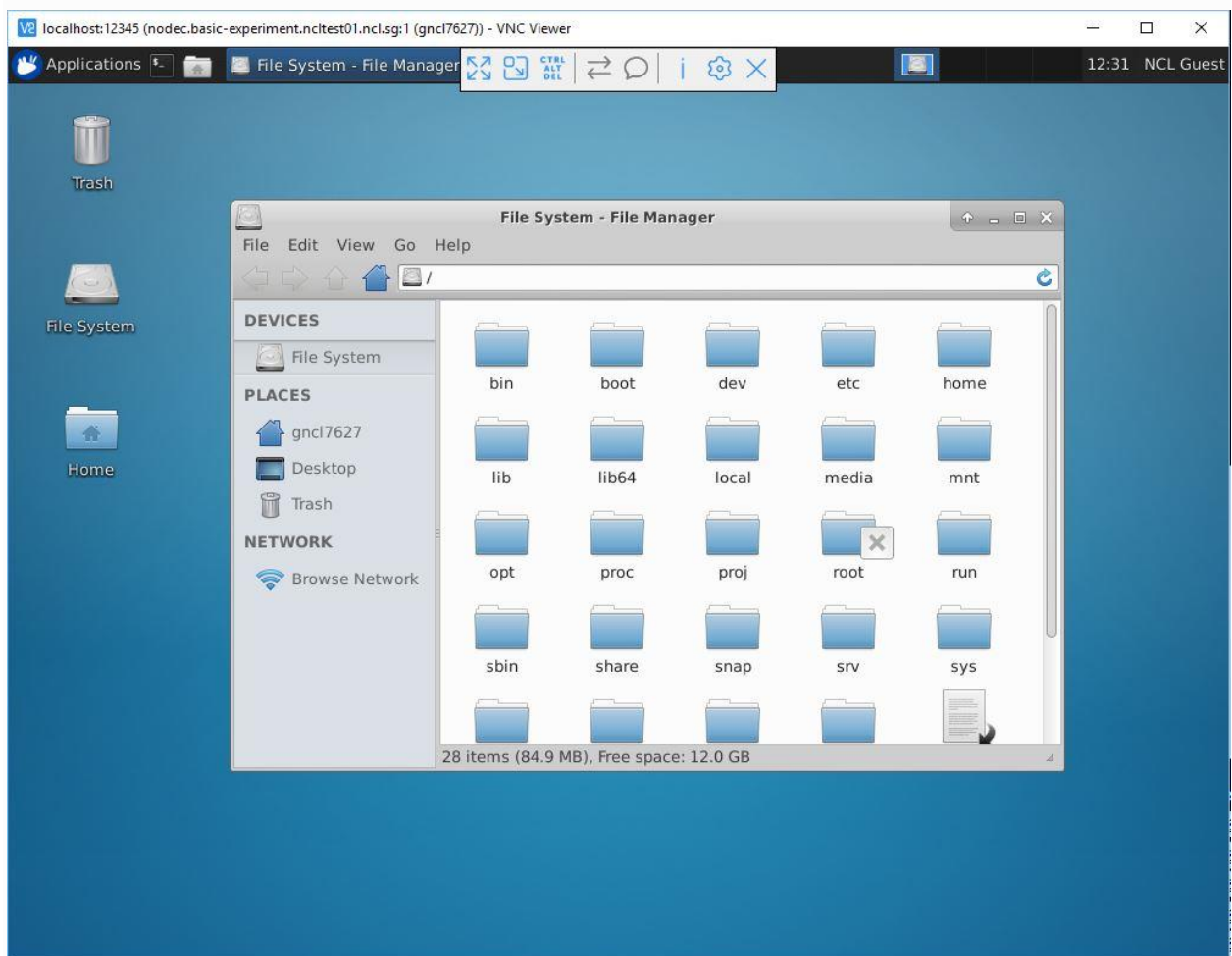
Now open that connection. You will see a login prompt, and you should log in to `users`.



Now you should be able to point your local VNC Viewer (one can be downloaded from [here](#)) to localhost port 12345, like below.



If the node is an Ubuntu node with desktop GUI installed, you will see something like this:

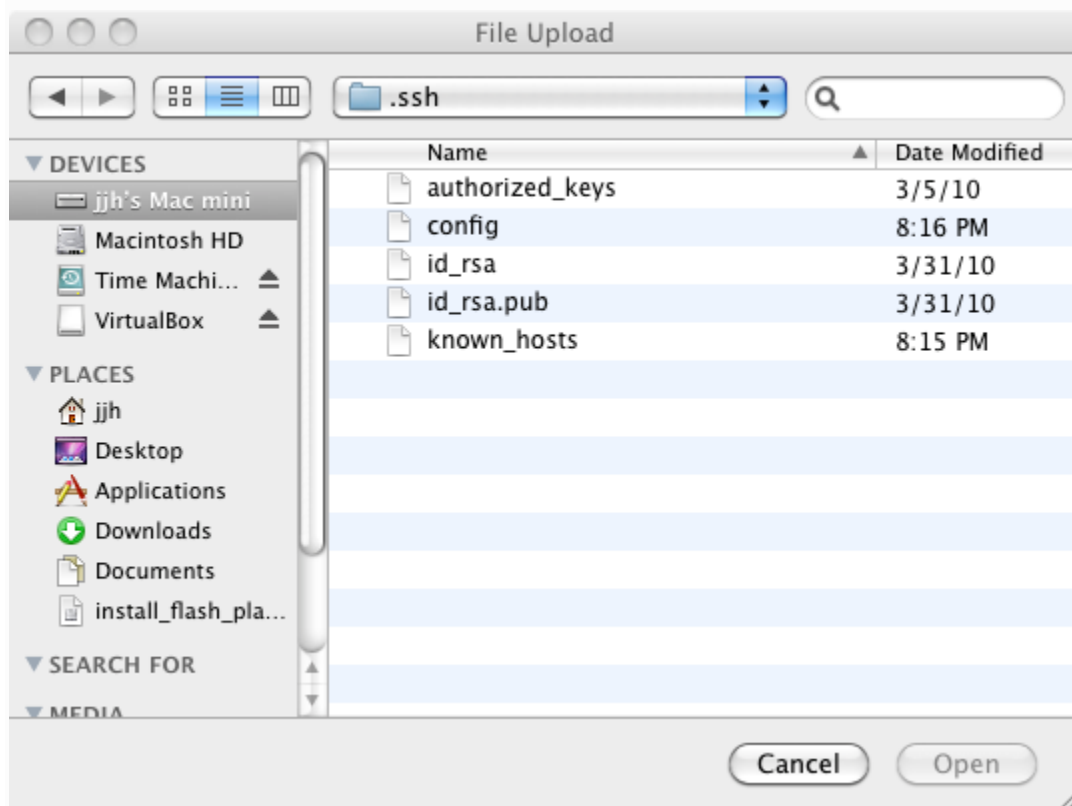


Be sure that your local machine does not firewall the local port 12345. Replace `pc22f` with a node in your experiment and the forwarded port with the port your service uses.

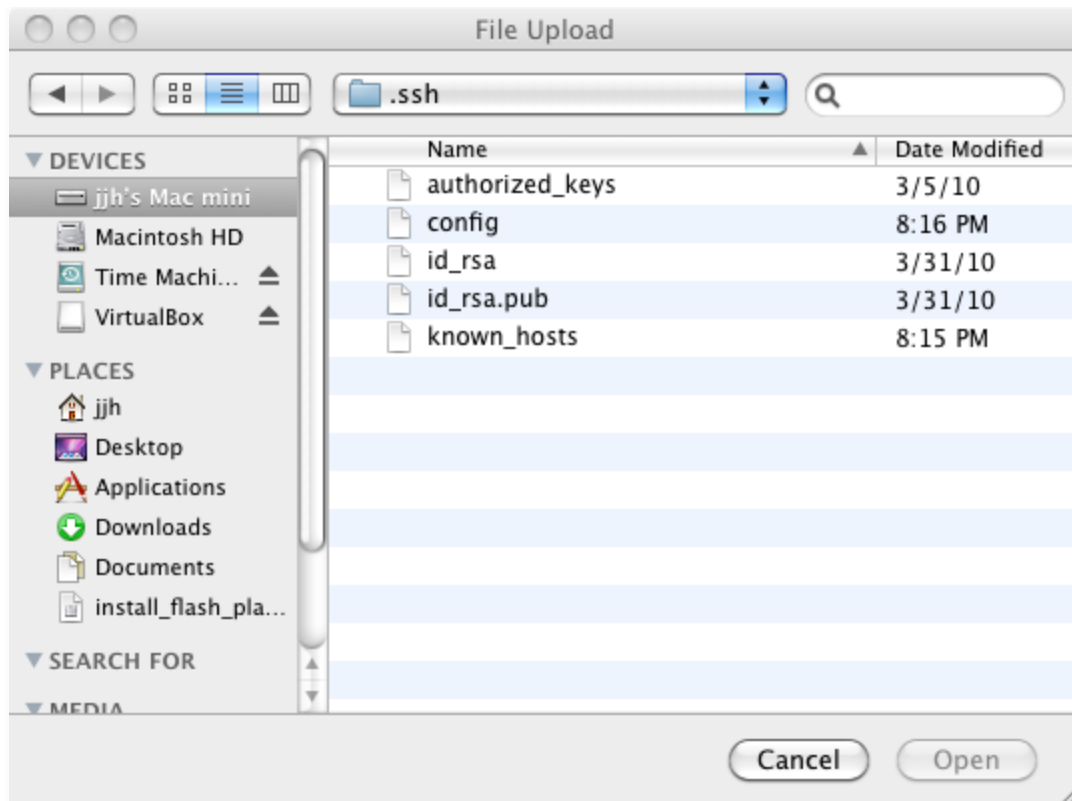
Uploading your SSH key from OS X

The Upload File dialog in Macintosh OS X does not show hidden directories by default. This creates an extra hassle when uploading SSH public keys from an OS X machine.

In the "Upload File" dialog, use the shortcut "Shift-Command-G" and type in "~/ssh" to navigate to the contents of your .ssh directory.



Then you will be presented with the contents of your .ssh directory and will be able to upload your `id_rsa.pub` file to NCL:



OpenSSH Configuration for Directly Logging into testbed nodes

These configuration tweaks should work for any operating system that runs OpenSSH (Linux, BSD, and OS X typically use OpenSSH as the default SSH client).

It is possible to log directly into testbed nodes with a little SSH configuration tweaking. Adding the following statement to "'~/.ssh/config'" will allow you to skip logging into users in order to access a particular testbed node. Change MyProject to the name of your project.

```
Host pc*.ncl.sg
    ProxyCommand ssh users.ncl.sg nc %h %p
    StrictHostKeyChecking no

Host *.MyProject.ncl.sg
    ProxyCommand ssh users.ncl.sg nc %h %p
    StrictHostKeyChecking no
```

With this configuration change and a proper SSH key setup, you will be able to directly log into nodes in your experiment.

You will now be able to log into nodes in your experiment using either the actual node name, e.g. pc10a.ncl.sg, or [host].[experiment].[project].ncl.sg.

For example:

```
~$ ssh gncl7627@nodeA.basic-experiment.ncltest01.ncl.sg
Password for gncl7627@users:
Warning: Permanently added 'nodea.basic-experiment.ncltest01.ncl.sg' (RSA) to
the list of known hosts.
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-40-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Jun 25 21:04:22 PDT 2018

System load:  0.0                Processes:            276
Usage of /:   13.0% of 15.62GB   Users logged in:     0
Memory usage: 0%                IP address for eth0: 10.16.0.52
Swap usage:   0%                IP address for eth5: 172.16.1.3

Graph this data and manage this system at:
https://landscape.canonical.com/

Last login: Mon Jun 25 21:04:22 2018 from 10.64.0.14
gncl7627@nodea:~$
```

Accelerating Multiple Connections using OpenSSH Connection Multiplexing

You can log in multiple times using the same SSH connection. This dramatically speeds up creating new connections. To enable SSH connection multiplexing, add the following lines to ~/.ssh/config. If you are on a multiuser machine, you may want to store the control socket someplace other than /tmp.

```
Host users.ncl.sg
    ControlMaster auto
```

```
ControlPath /tmp/%r@%h:%p
```

To verify that it is working, you can use the `"-v"` option:

```
~$ ssh -v pc026.ncl.sg
OpenSSH_5.2p1, OpenSSL 0.9.8l 5 Nov 2009
debug1: Reading configuration data /Users/jjh/.ssh/config
debug1: Applying options for *ncl.sg
debug1: Applying options for pc*.ncl.sg
debug1: Applying options for *
debug1: Reading configuration data /etc/ssh_config
debug1: auto-mux: Trying existing master
Last login: Wed Nov 10 20:51:43 2010 from users.ncl.sg
node0:~>
```

If you try to close your master connection while other connections are active, the connection will stay running until the other sessions end.

Node Types

This is not a complete list of all node types available at NCL Testbed, but below are the primary types.

<https://ncl.sg/testbedNodesStatus> (SystemX and Intel)

<https://ncl.sg/testbedInformation> (Node Specifications)

Operating System Images

Here is the list of currently supported NCL Testbed operating system images. If you have a NCL Testbed account, you can view the most updated information on the [testbed info page](#) on the web interface.

Supported OS Images as of 06/30/2018

| Name | OS | Description |
|-------------------|-------|---|
| Ubuntu1404-64-STD | Linux | Ubuntu 14.04 LTS 64 bit Standard Image |
| Ubuntu1604-64-STD | Linux | Ubuntu 16.04 LTS 64 bit Standard Image |
| Ubuntu1604-64-GUI | Linux | Ubuntu 16.04 LTS 64 bit Image with Desktop GUI and VNC Server |

Creating Custom Operating System Images

What is an Operating System ID (OSID) versus an Image ID?

In order to make the best use of custom operating system images, it is important to understand the difference between these two concepts.

- An **Image ID** is a descriptor for a disk image. This can be an image of an entire disk, a partition of a disk, or a set of partitions of a disk. By supporting multiple partitions, we can technically support different operating systems within the same disk image. These are referred to as *combo images*. The Image ID points to a real file that is stored in the directory `/proj/YourProjectName/images`. Other critical information is associated with the Image ID, such as what node types are supported by the images and what operating systems are on each partition. You can view the Image IDs on the [Teams page](#) (Your teams' saved OS images) on the testbed web interface.
- An **OSID** describes an operating system which resides on a partition of a disk image. Every Image ID will have at least one OSID associated with it. In typical testbed usage, the Image ID and OSID will be the same since we usually do not put multiple operating systems on a single image.

Standard Testbed Images

We provide a number of supported testbed images here at NCL Testbed. These images can be viewed by looking at the [List of Standard OS Images on the Testbed Info page](#).

The supported testbed images are listed in the [Operating System Images](#) documentation page.

Custom OS Images

If your set of operating system customizations cannot be easily contained within an RPM/TAR (or multiple RPM/TARs), then you can create your own custom OS image. NCL Testbed allows you to create your own disk images and load them on your experimental nodes automatically when your experiment is created or swapped in.

Once you have created a custom disk image, you can use that image name in your NS file. When your experiment is swapped in, the testbed system will arrange for your disks to be loaded in parallel using a locally written multicast disk loading protocol.

Note

Experience has shown that it is much faster to load a disk image on 10 nodes at once, then it is to load a bunch of RPMS or tarballs on each node as it boots. So while it may seem like overkill to create your own disk image, we can assure you it is not.

The most common approach is to use the [Save Image](#) form to create a disk image that contains a customized version of a standard Linux or the FreeBSD image. All you need to do is enter the image name in the form, and the testbed system will create the image for you automatically, notifying you when it is finished. You can then use that image in subsequent experiments by specifying the image name in your NS file with the `tb-set-node-os` command. When the experiment is configured, the proper image will be loaded on each node automatically by the system. Please refer to [this tutorial](#) for more details.

Creating Your Custom Image

A typical approach to creating your own disk image is using one of the default images as a base or template. To do this:

1. Create a single-node Linux or FreeBSD experiment. In your NS file, use the appropriate `tb-set-node-os` command, as in the following example:

```
tb-set-node-os $nodeA Ubuntu1404-64-STD
```

2. After your experiment has swapped in (the experiment status changed to running), log into the node and load all of the software packages that you wish to load. If you want to install the latest version of the Linux kernel on one of our standard disk images, or on your own custom Linux image, be sure to arrange for any programs that need to be started at boot time. It is a good idea to reboot the node and make sure that everything is running as expected when it comes up.
3. Note the physical (`pcXXX`) name of the machine used!
4. Create an image using the Save Image form (Experiments page -> View Realization -> Save this node's image).
5. Wait for the image status changed to "Ready" (Teams page -> Your teams' saved OS images).
6. Now you can create a second single-node experiment to test your new image. In your NS file, use `tb-set-node-os` to specify the image that you just created. Be sure to remove any RPM or tarball directives. Submit that NS file and wait for the experiment to start. Then log into the new node and check to make sure everything is running normally.
7. If everything is going well, terminate both of these single-node experiments. If not, release the experiment created in the previous step, and then go back and fix the original node (`pcXXX` above). Recreate the image as needed:

```
create_image -p <proj> <imageid> <node>
```

8. Once your image is working properly, you can use it in any NS file by using the `tb-set-node-os` command. If you ever want to reload a node in your experiment, either with one of your images or with one of the default images, you can use the `os_load` command. Log into `users` and run:

```
os_load -p <proj> -i <imageid> <node>
```

This program will run in the foreground, waiting until the image has been loaded. At that point you should log in and make sure everything is working okay. If you want to load the default image, then simply run:

```
os_load <node>
```

Hints When Making New OS Images

1. Consider creating a **two** node experiment, one to create the image and the other to load it back.

There is a command called `os_load` available on the `users` server:

```
users% which os_load
/usr/testbed/bin/os_load
users% os_load -h
option -h not recognized
os_load [options] node [node ...]
os_load [options] -e pid,eid
where:
    -i    Specify image name; otherwise load default image
    -p    Specify project for finding image name (-i)
    -s    Do *not* wait for nodes to finish reloading
    -m    Specify internal image id (instead of -i and -p)
    -r    Do *not* reboot nodes; do that yourself
    -e    Reboot all nodes in an experiment
    node  Node to reboot (pcXXX)
```

2. If you think you've got a good image, but it flounders while coming up, create another experiment with an NS directive that says *"Even if you think the node has not booted, let my experiment swap in anyway."* This may allow you to log in through the console and figure out what went wrong. An example of such a directive is:

```
tb-set-node-failure-action $nodeA "nonfatal"
```

3. Create whole disk images on a smaller machine rather than a single partition image.

Guidelines for Students

- Read [Student Introduction to NCL Testbed](#) - especially the [Things to keep in mind](#) section.
- Read [User DOs and DON'Ts](#).
- Contact your TA or instructor first with NCL Testbed problems. They can help you with password reset, inability to log on, inability to swap in and many other problems. They will pass any they cannot solve to the Testbed Ops team. Do NOT contact Ops directly.
- Pace yourself and do not leave work for the last day before the deadline. Many courses share the testbed, along with many researchers. **There may not be enough resources for you if you ask for them at the last moment.**
- Promptly swap out experiments if you will not use them for at least an hour. Read [Student Introduction to NCL Testbed](#) to learn how to save your work and retrieve it on the next swap in.
- If you cannot swap in due to lack of free machines, keep trying for a day. Our load goes down during nights and weekends. If you still have problems after a day contact your instructor or TA who will request help from testbed ops.

Student Introduction to NCL Testbed

What is NCL Testbed?

The NCL Testbed is a nationally shared testbed that provides computing resources, repeatable and controllable experimentation environments, as well as application services. The testbed includes a cluster of 200 nodes that provides a wide range of provisioning mechanisms, security data and security services. NCL aims to provide a platform that fosters and encourages collaboration among researchers in academia, government bodies and the industry.

The software stack that manages the NCL testbed is based on [DETERLab](#), which is a security and education-enhanced version of [Emulab](#). NCL is also collaborating with the DETERLab team on [testbed research](#).

The NCL lab is funded by the National Research Foundation (NRF) since November 2015.

NCL Testbed is a shared testbed providing a platform for research in cyber security and serving a broad user community, including academia, industry, and government. To date, NCL Testbed-based projects have included behavior analysis and defensive technologies including DDoS attacks, worm and botnet attacks, encryption, pattern detection, and intrusion-tolerant storage protocols."

NCL Testbed (like Emulab) offers user accounts with assorted permissions associated with different experiment groups. Each group can have its own preconfigured experimental environments running on Linux, BSD, Windows, or other operating systems. Users running NCL Testbed experiments have full control of real hardware and networks running preconfigured software packages. These features make it an ideal platform for computer science and especially computer security education. Many instructors have designed class exercises (homework assignments, project assignments, in-class demos, etc.) consisting of a lab manual, software, data, network configurations, and machines from NCL's pool. This allows each student to run her own experiments on dedicated hardware.

How does it work?

The software running NCL Testbed will load operating system images (low level disk copies) onto to free nodes in the testbed, and then reconfigure programmable switches to create VLANs with the newly-imaged nodes connected according to the topology specified by the experiment creator. After the system is fully imaged and configured, NCL Testbed will execute specified scripts, unpack tarballs, and/or install rpm files according to the experiment's configuration. The end result is a live network of real machines, accessible via the Internet.

How do I get a NCL Testbed login?

1. Your instructor will request an account for you. Simply send your preferred email address to your instructor.
2. Once the testbed ops set up your account, you will receive an email with your username and password at the address you supplied.
3. Within one week, use those credentials to log in.
4. Edit your profile as follows: a. Click user icon on the top-right corner. b. Choose "Account Settings" menu option. c. Replace any default contents in the fields shown in the Account Info form. d. Click "Save settings".

Using NCL Testbed

How do I start an exercise?

Before you can perform the tasks described in your exercise assignment, you will, in many cases, need to create an experiment in NCL Testbed to work on. This will be your environment to use whenever you need it. To create a new experiment, follow the instructions [here](#) and tutorial [here](#).

How do I work on my exercise?

Locate the experiment you just created on the Experiments page, click "Start this experiment" then click "Confirm". The starting / swap in process will take about 10 minutes. While you're waiting, you can watch the swap in process displayed in your web browser. When the experiment has finished swapping in, you can perform the tasks in your exercise manual.

How do I access my experiment?

Your experiment is made up of one or more machines on the internal NCL Testbed network, which is behind a firewall. To access your experimental nodes, you'll need to first SSH to `users.ncl.sg`. If you don't know how to use SSH, see our [tutorial](#).

`users.ncl.sg` (or `users` for short) is the "control server" for NCL Testbed. From `users` you can contact all your nodes, reboot them, etc.

Once you log in to `users`, you'll need to SSH again to your actual experimental nodes. Since your nodes' addresses may change every time you swap them in, it's best to SSH to the permanent network names of the nodes. Documentation [here](#) and tutorial [here](#) show you how to access your nodes using SSH.

Congratulations! Your lab environment is now set up, and you can get to work at the tasks in your lab manual. Make sure you read the "Things to keep in mind" section below!

Some labs benefit from Port Forwarding. Port Forwarding is a technique that can allow you to access your experimental nodes directly from your desktop computer. This is especially useful for accessing web applications running on your experimental nodes. See our [SSH tutorial](#) for more information.

Finally, when you are done working with your nodes, you should save your work and swap out the experiment so that someone else can use the physical machines.

Things to keep in mind

Saving and securing your files on NCL Testbed

Every user on NCL Testbed has a home directory on `users.ncl.sg` which is mounted via NFS (Network File System) to experimental nodes. This means that anything you place in your home directory on one experimental node (or the `users` machine) is visible in your home directory on your other experimental nodes. Your home directory is private, so you may save your work in that directory. However, everything else on experimental nodes is permanently lost when an experiment is swapped out.

Make sure you save your work in your home directory before swapping out your experiment"

Another place to save your files would be `/proj/YourProject`. This directory is also NFS-mounted to all experimental nodes so same rules apply about writing to it a lot, as for your home directory. It is shared by all members of your project/class.

NOTE: There is a disk quota limit on how many files you can store on your home or project directory. The current limit is 25GB. If you have reached that limit, you need to clear files in your home directory or project directory, or move files to your nodes' local storage (see [I need more disk space on my node](#)).

Again, on NCL Testbed, files ARE NOT SAVED between swap-ins. Additionally, class experiments may be forcibly swapped out after a certain number of idle hours (or some maximum amount of time).

You must manually save copies of any files you want to keep in your home directory. Any files left elsewhere on the experimental nodes will be erased and lost forever. This means that if you want to store progress for a lab and come back to it later, you will need to put it in your home directory before swapping out the experiment.

Swap out -- DON'T "terminate"!

When you are done with your experiment for the time being, please make sure you save your work into an appropriate location and then swap out your experiment. To do this, use the "Stop

this experiment" button on the Experiments page. This allows the resources to be deallocated so that someone else can use them.

Do not use the "Remove experiment" button unless you are completely finished with your exercise. Termination (or removing) will erase the experiment and you won't be able to swap it back in without recreating it.

Swapping out is the equivalent of temporarily stopping the experiment and relinquishing the testbed resources. Swapping out is what you want to do when you're taking a break from the work, but coming back later. Terminating says "I won't need this experiment again, ever." This may be confusing, especially since "Swap Out" seems to imply that it saves your progress (it doesn't, as described above). Just remember to Swap In/Out, and never "Terminate" unless you're sure you're completely done with the experiment. If you do end up terminating an experiment, you can always go back and recreate it.

Submitting your work to your instructor

Each exercise manual has a section entitled "Submission Instructions," and your instructor may have given you additional instructions for submission. Follow the instructions in that section, and submit your work to your instructor.

Unless otherwise instructed, it's a good idea to include:

Your name Your preferred email address Your student ID (if applicable) Your NCL Testbed username Your experiment's name (e.g., jstudent-exploits)