Planetlab Project How-to

By Danny Bickson
Daniel51@cs.huji.ac.il
DANSS LAB (Distributed Algorithms Networking and Secure Systems Group),
The Hebrew University of Jerusalem
http://www.cs.huji.ac.il/labs/danss

7.10.03	Version 1.0	Initial document.
14.10.03	Version 1.1	Incorporated sergs@cs remarks regarding ssh.
9.1.04	Version 1.2	Added ssh help
13.1.04	Version 1.3	Added rpm support
12.2.04	Version 1.4	Added ScriptRoute doc by Tal Haramaty – tal128@cs
14.9.04	Version 1.5	Added ssh tips
17.11.04	Version 1.6	Added "expect", Java example
24.7.05	Version 2.0	Revised document, including v. 3 support, strace

Introduction

Planetlab is a distributed testbed for running distributed software. Planetlab home page is found on: http://www.planet-lab.org

Here at the DANSS lab we currently host 5 planetlab nodes. We run several experiments using the Planetlab testbed. Our Planetlab research web page is found on: http://www.cs.huji.ac.il/labs/danss/research.html#planet

The PI (principle investigator) is Prof. Danny Dolev (dolev@cs). The site administrator is Danny Bickson (daniel51@cs).

New Users

1. In order to be able to login into Planetlab computers, you should first register into the Planetlab joining users page: http://www.planet-lab.org/php/join_user.php
   Our site name is: The Hebrew University of Jerusalem. You are NOT a PI. (See section 3).
2. For creating a SSH private/public key pair, use the ssh-keygen program on one of the Linux machines (like mangal, sands):

   ```
   ssh-keygen -t rsa
   ```

   `ssh-keygen` asks for a passphrase. A passphrase is similar to a password, except it can be up to 30 characters long. The output of the `ssh-keygen` command is two files: a private key named `id_rsa` and a public key named `id_rsa.pub`. Both files are generate at default in the directory `.ssh/` on your home directory.

Note: the optional `-f` flag is the name of the output file. You can change it in case you need to work with more then one ssh configurations.

3. After registration you should email me daniel51@cs.huji.ac.il and cc Prof. Danny Dolev (dolev@cs) which is the DANSS site PI asking to activate your account. You will get in a response, a slice name (in the format huji_<username>) for your usage.
4. Now you can login into your Planetlab account in order to verify your details. There is a login link on the bottom of left frame in the Planetlab home page. After logging in, you can view your account information by clicking the "View Account" link. You should see something like:

Danny Bickson

First Name	Danny
Last Name	Bickson
Title	DANSS System Administrator
Is PI	f
E-mail	daniel51@cs.huji.ac.il
URL	http://www.cs.huji.ac.il/labs/danss
…	…
…	…
Site Name	The Hebrew University of Jerusalem
Enabled	t
Admin Priv	f
SSH PubKey	**[Download]**

Check that the account is enabled.

Planetlab machines v. 2 where based on Red Hat Linux 9.3. V. 3 machines are based on Fedora Core 2. Each user (user is named slice) has a virtual environment where he can see only his own processes files etc. The superuser (root) has limited functionality. Each slice can work as a root by using the "su" command (without any password).

Logging into Planetlab machines

On any Linux machine which is owned by the system (like mangal, inferno, gx-##, xil-##, bmos-## etc) run the following command:

```
ssh -l <slicename> <machinename>
```

For example:

```
<1|1>daniel51@mangal:~> ssh -l huji6 planet1.cs.huji.ac.il
Enter passphrase for key '/cs/grad/daniel51/.ssh/id_rsa':
Last login: Mon Oct  6 14:34:48 2003 from mangal.cs.huji.ac.il
[huji6@planet1 huji6]$
```

Executing a command

You can use ssh for executing a command (using a non-interactive mode).
Add the command argument as the last argument for the ssh command.

```
ssh –l <slicename> <machinename> <command>
```

For example:

```
<204|0>daniel51@mangal:~> ssh -l huji_anothershura
planet2.cs.huji.ac.il pwd
```

And the result:

```
/home/huji_anothershura
```

Using ssh-agent

Each time you will login using ssh, you will be asked for your passphrase. If you want to enable a login session without entering the passphrase each time you should write:

```
eval `ssh-agent`
ssh-add
```

```
<4|1>daniel51@mangal:~>eval `ssh-agent`
Agent pid 11693
<4|1>daniel51@mangal:~> ssh-add
Enter passphrase for /cs/grad/daniel51/.ssh/id_rsa:
Identity added: /cs/grad/daniel51/.ssh/id_rsa
(/cs/grad/daniel51/.ssh/id_rsa)
```

For the rest of this session you will not be prompted for passphrase.

After you are logged in using ssh, you can work as on any Red Hat Linux machine. (You can switch shells, install packages etc.)

Other SSH Tips

Running in batch mode

For the first time you will connect to any remote machine using ssh you will get the following warning:

```
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)?
```

That is because new host keys are added at the first connection to `~/.ssh/known_hosts` file.

In case you have an host key which is not updated, you will get the following error:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: HOST IDENTIFICATION HAS CHANGED!         @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

```
Someone could be eavesdropping on you right now
(man-in-the-middle attack)!
It is also possible that the host key has just been changed.
Please contact your system administrator.
Add correct host key to "/home/name/.ssh2/hostkeys/key_22_machine.pub"
to get rid of this message.
Received server key's fingerprint: xopif-fycak-pepap-lopuv-dyhov-fedas-liguz-nuvic-gubut-
kigan-nixix
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Agent forwarding is disabled to avoid attacks by corrupted servers.
X11 forwarding is disabled to avoid attacks by corrupted servers.
Are you sure you want to continue connecting (yes/no)?
```

One solution is to delete the offending line of the `~/.ssh/known_hosts` file which contains the stale key.

In order to bypass the above ssh questions:

Create a file named `~/.ssh/config` in your home dir containing the following lines:

```
StrictHostKeyChecking no
BatchMode yes
```

All the annoying questions will be answered affirmatively.

The Planetlab `known_hosts` list can be found at:
http://lists.planet-lab.org/pipermail/support-sf/2003-August/002734.html
You can download it and override you own file.

Installing software on a remote machine using SSH

When you want to install a program like yum on a planetlab machine you have to be root. But how to install it on many machines? try to ssh using the command
`su -c 'command to run as root'`
This command will **not** work unless you will specify the flags:

```
ssh -n -t -l <slice name> <machine name> "su -c 'command'"
```

This command will not work in the background (using shell "&")

Using expect instead of using the ssh-agent

Another option for login is using the "`expect`" Linux utility.
Attached is an example script which is taken from PlanetLab tools.

```
#!/usr/bin/expect -f

#This Expect script can ssh to a remote host without typing
#a password or passphrase.
#
# Author:  Jianping Wang, 2003
```

```
#       Multimedia Networks Group, University of Virginia
#
#       SOURCE CODE RELEASED TO THE PUBLIC DOMAIN
#
#       version 1.0 - 04/01/2003
# Modifications to script by Danny Bickson, 2004
# you should fill in the fields UserId, Passphrase and remotehost

set UserID ""
set Passphrase ""
set remotehost ""

spawn ssh -l $UserID $remotehost
expect -re "Enter passphrase for key '.*':"
send "$Passphrase\r"
interact
```

Avoiding ssh timeouts – using ssh

When trying to connect a machine which is down, ssh might get stuck for a certain time. For avoiding this, you can change the "ConnectionTimeout" parameter in the .ssh/config file to a desired timeout in seconds.

Avoiding ssh timeouts – using perl script

This script will interrupt a stuck ssh session after X seconds (currently 10 seconds in the example). Create the following perl script in a file named `timeout.pl`

```
#!/usr/local/bin/perl -w
my $res = -1;
eval {
        my $command = $ARGV[0];
        local $SIG{ALRM} = sub { exit 1; };
        alarm(10);
        $res=system($command);
        alarm(0);
};

if (($res != 0 ))
        { exit 1; }
else { exit 0; };
```

Then you can run ssh using "timeout.pl <ssh command>"

Other SSH resources (taken from PL mailing list)

This document has a good overview of the most common uses of ssh:
http://kimmo.suominen.com/ssh/

Also, if you haven't seen this document, it outlines some basic points on using ssh keys on PlanetLab:
https://www.planet-lab.org/db/web_accounts/ssh_public_key.php

Most linux systems will have ssh (OpenSSH) already installed for you, but if this is not the case, visit http://openssh.com/ for official releases for numerous linux-based platforms.

If you are using Windows based machines, several options exist. There are commercial ssh clients like SecureCRT:
http://www.vandyke.com/products/securecrt/
or freeware clients like putty:
http://www.chiark.greenend.org.uk/~sgtatham/putty/

Another option worth mentioning is cygwin. For windows, its a complete linux like environment. For more information, see: http://cygwin.com/
If you install it, make sure to add OpenSSH to the list of packages to install, as I last checked, it was not enabled by default. Once installed, this will give you a command prompt running the common bash shell, and you can use ssh as if you were on a linux box.

For information on how to setup ssh-agent:
http://kimmo.suominen.com/ssh/#ssh-agent
http://www.caip.rutgers.edu/~vincentm/LINKS/sshagent.html

For windows, there is a similar program available from the same authors who wrote putty, called Pageant. This works in the same way, where you add your keys to the Pageant application after its running, and putty contacts Pageant for the authorization while logging in. See the putty website above for more specific documentation.

Copying files to/from Planetlab machines

Using SCP to copy files from/to a PlanetLab machine

This section was taken from:
http://sc.tamu.edu/help/general/accessMethods/openssh_unix.html

Use SCP to copy files from one machine to another. SCP replaces rcp and should be used instead of ftp. It also has more flexibility than ftp and can be used to copy directories instead of just files. The general form of SCP is:

```
scp [[user@]host1:]filename1 [[user@]host2:]filename2
```

Where `filename1` and `filename2` can be file or directory names. If your user name is the same on both the local and remote machines, then you do not have to provide the `user@`. If you are copying from your local machine, you do not have to provide the name of host1. For example, to copy a file called temp.ps from a local machine to agave.tamu.edu, use the following command:

```
scp temp.ps agave.tamu.edu:temp.ps
```

The file `temp.ps` will be copied to your home directory on agave. If you have a different user name on agave, then specify it as shown below.

```
scp temp.ps remote-user-name@agave.tamu.edu:temp.ps
```

You can use the "-r" flag to recursively copy subdirectories:

```
scp –r temp.ps remote-user-name@agave.tamu.edu:temp.ps
```

```
Don't forget to
```

Using rsync to copy a directory tree from/to Planetlab machine

`rsync` is a utility to synchronize two directories to the same content. It can be used instead of scp for copying files into Planetlab machines. In order to operate, you should use "`rsync –e ssh`" for working over ssh in Planetlab.
Rsync example: (copies the full content of the directory `~daniel51/Planetlab/cog1/` including subdirectories into the remote directory `PL/Cogs/cog1` on the remote machine `planet1.cs.huji.ac.il`):

```
rsync -rvaz -e ssh --progress --delete ~daniel51/Planetlab/cog1/
PL/Cogs/cog1 planet1.cs.huji.ac.il
```

Instead of using the above "`rsync –e`" flag you can use:
`setenv RSYNC_RSH ssh`
Make sure `RSYNC_SSH` is defined on every XSession. Use this option on cases the "-e" flag is not working.

Possible rsync error message

If you are getting a similar error:

```
> unexpected tag 88
> rsync error: error in rsync protocol data stream (code 12) at
io.c(298)
```

Try to use rsync with the `"--blocking-io"` option.

Installing packages

Since the Planetlab installation is minimal, applications like man, make, gzip , tar etc. are not installed. For installing additional RPMs you can use one of the following:

Installing packages using yum

```
curl http://boot.planet-lab.org/alpina/other-scripts/setup_yum.sh | bash
```

```
yum groupinstall "Development Tools"
```

You may also want:

```
yum -y install man gzip less gcc flex bison bc rpm-build
```

Installing packages using apt-get

You can install any rpms you want. Use apt-get (freshrpms.net) or yum may help the installation process. E.g., if you want "make", do:

```
su -
rpm -Uvh http://boot.planet-lab.org/install-rpms/stock-rh9/apt-get-xxx.rpm

apt-get update


apt-get update
apt-get install make
```

Compiling your software on PlanetLab

It is not recommended to compile your software on a PlanetLab machine, since the machines are already heavily loaded. You can compile your software on a local Fedora core 2 machine, and get binaries which are 99% compatible to PlanetLab.

Non-recommended (but possible) option using make

Technically, you can compile on a Planetlab machine using "make". You first need to install make, gcc or other packages you need to compile:
`yum -y install make gcc`
And then compile your software as usual.

A better option: install a local PlanetLab node

You will first need to install a fedora core 2 machine. Then you will need to upgrade it to the Planetlab software using the instructions on
https://wiki.planet-lab.org/twiki/bin/view/Planetlab/DevBox

Misc Tips

Intalling Java

Example script from "Nelson A. da Nۥbrega Jr." [nelson@dsc.ufcg.edu.br]

Filename: installconfig_java.txt

```
file=$1
j2re=$2
installjava=$3
echo Starting instalation...
echo ========================
for node in `cat $file`
do
  echo ========================
  echo $node
```

```
  echo Copying J2RE to $node
  scp $j2re ufmg_09@$node:.
  echo Copying script installjava
  scp $installjava ufmg_09@$node:.
  echo Entering at $node
  ssh -x ufmg_09@$node "echo Installing J2RE at $node; ./$installjava $j2re"
  echo ========================
done
```

Filename: installjava.txt

```
j2re=$1
echo Installing java.bin
echo "yes" > yes
./$j2re < yes
rm yes
echo Rename dir j2re1.4.2_05/ to java/
mv j2re1.4.2_05/ java/

echo Setting Environments variables
echo "CLASSPATH=.:\$HOME/java/lib
export CLASSPATH

JAVA_HOME=\$HOME/java
export JAVA_HOME

JDK_HOME=\$JAVA_HOME
export JDK_HOME

PATH=.:\$PATH:\$JAVA_HOME/bin
export PATH" >> .bashrc

source .bashrc
```

Operating your software on Planetlab

Most (if not all..) of the experiments on Planetlab are communications related. The Planetlab v. 3 is using virtualized network access. http://www.planet-lab.org/PDN/PDN-05-029/

Some of your applications, or 3rd party applications you are using might not work correctly since Planetlab allows only a subset of network protocols: TCP, UDP, ICMP , GRE and PPTP.

Most of the errors are because in Planetlab the applications needs to `bind`() the socket before usage in order to relate it to a certain slice.

Debugging your programs

Using strace utility

In case your program works on a regular Linux but not on Planetlab, you can use "`strace`" in order to debug your application. Only add “strace” before your full command.

For example: `Nettimer` utility did not run with error of

```
nettimer: transport.c:530: transport_send_packet: Assertion
>`(*__errno_location ()) == 90' failed.
```

Using strace produces the following log file:

…

```
fstat64(4, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x123000
read(4, "Iface\tDestination\tGateway \tFlags"..., 1024) = 512
read(4, "", 1024)                       = 0
close(4)                                = 0
munmap(0x123000, 4096)                  = 0
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = -1 EPERM (Operation not permitted)
write(2, "KLException caught in program: \""..., 310KLException caught in program:
"/usr/sbin/nettimer"  at file: "active_probing.c" function: "active_probing_new" line:
211   Thrown at file: "transport.c" function: "session->socket = socket(AF_INET,
SOCK_RAW, IPPROTO_RAW)" line: 396
meaning: "System error: 'Operation not permitted' Could not open socket"
) = 310
exit_group(1)                           = ?
```

It is clear the opening socket using the SOCK_RAW protocol failed. That is because it was not run using in superuser mode. After running as root, Nettimer produced the following error: `transport.c:530: transport_send_packet: Assertion `(*__errno_location ()) == 90' failed.`

Running again with strace produced the following log:

```
rt_sigprocmask(SIG_BLOCK, ~[RTMIN], [], 8) = 0 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) =
0
close(6)                                = 0
rt_sigprocmask(SIG_BLOCK, ~[RTMIN], [], 8) = 0 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) =
0
close(5)                                = 0
sendmsg(4, {msg_name(16)={sa_family=AF_INET, sin_port=htons(10100),
sin_addr=inet_addr("128.112.139.102")},
msg_iov(1)=[{"E\0\1\364\0\0\0\0\377\6\337!\204\343J(\200p\213f\327\271"...,
500}], msg_controllen=0, msg_flags=0}, 0) = -1 EPERM (Operation not
permitted)
write(2, "nettimer: transport.c:530: trans"..., 99nettimer: transport.c:530:
transport_send_packet: Assertion `(*__errno_location ()) == 90' failed. ) = 99
rt_sigprocmask(SIG_UNBLOCK, [ABRT], NULL, 8) = 0
tgkill(17232, 17232, SIGABRT)           = 0
--- SIGABRT (Aborted) @ 0 (0) ---
+++ killed by SIGABRT +++
```

Here the sendmsg() function call failed, probably because the socket was not bound before sending.

Using tcpdump utility

You can use tcpdump utility for viewing your slice network traffic. The command line is:
`sudo /usr/sbin/tcpdump -i vnet`

Using tethereal utility

Command syntax:
`tethereal -i vnet`
Example of a capture done while running `"curl http://www.yahoo.com/"`:

```
   2.845346 128.112.136.10 -> 128.112.139.71 DNS Standard query response
PTR p25.www.re2.yahoo.com
   2.846096 128.112.139.71 -> 68.142.226.37 TCP 51850 > http [SYN] Seq=0
Ack=0 Win=5840 Len=0 MSS=1460 TSV=1773606008 TSER=0
   2.863092 128.112.139.71 -> 68.142.226.37 TCP 51850 > http [ACK] Seq=1
Ack=0 Win=5840 Len=0 TSV=1773606025 TSER=135373488
   2.863029 68.142.226.37 -> 128.112.139.71 TCP http > 51850 [SYN, ACK]
Seq=0 Ack=1 Win=1460 Len=0 MSS=1460 TSV=135373488 TSER=1773606008
   2.863739 128.112.139.71 -> 68.142.226.37 HTTP GET / HTTP/1.1
```

We can see in the trace the DNS response and the TCP connection handshake. Thanks to Mark Huang (PLC Princeton).

Creating a core file

In case your application crashes on a Planetlab node and you want to debug it.
Before running the application, on a bash shell:

```
ulimit -c unlimited
```

On a csh/tcsh:

```
limit coredumpsize unlimited
```

Example of running using ssh:

```
ssh -l huji_daniel51 planetlab01.ethz.ch ulimit -c unlimited; \
                                          cd PL/Cogs/daniel51/; \
                                          ./client
```

If the program crashes, a core file should be created. You can debug it using

```
gdb client core
```

You might need to install gdb using yum, or copy the core file to a devbox machine and debug it there.

Working with the slice deploy utility

SliceDeploy utility is a set of Perl scripts which enable working from a centralized Linux machine, sending and receiving files to selected Planetlab machines and running commands on those machines, and getting the program output (logs) back.

On the Planetlab homepage, click the software link and search for the ”SliceDeploy” project. http://www.planet-lab.org/php/contrib/contrib.php From there, there is a link to download the scripts. Download the tar file and save it on your Linux home drive. Extract the tar file.

Inside the package there are two short documents: HOWTO and README. You are advised to print and read both. Start from HOWTO since it is more basic.

The main script is “sd” (SliceDeploy).

There are two small modifications to the sd script:

1. In order to setup the working directories, change the constants DISTDIR ,LIBDIR and BINDIR inside the “sd” script to your working directories where you opened the SliceDeply tar.
For example:

```
my $DISTDIR=".";
my $BINDIR="$DISTDIR/bin/";
my $LIBDIR="$DISTDIR/lib/";
```

Those changes are instead of running the “`INSTALL.sh`” file.

More on dynamic slices architecture:

www.cs.princeton.edu/~llp/slices.pdf

A short tutorial for using "sd":

(The HOWTO and README files have more details)

1. Create directory structure by using:
   `./sd createslice --slice <slicename>`
   A directory called "Sliver" is created. Anything you put into it will be copied into the Planetlab machines when you deploy.
2. Create subprojects by using
   `./sd createcog --cog <cogname>`
   (This is useful if you want to divide the project into several subprojects. If not, you can just create 1 default cog).
   A directory named Cogs/<cogname> will be created. Inside you will find a script called "PLCogCntl". You can add code in the sections START, STOP, DEPLOY, etc. If you want to run a script named "mysciprt.sh" remotely you add something like:

```
# ------------------------------------------------------
# START
# Start this service running
# ------------------------------------------------------
sub cSTART {
    # Your service code here
  &DoSystem("myscript.sh");
  print "-----Cog command START on .$ENV{"HOSTNAME"}."\n";
}
```

3. Add PlanetLab nodes into your project. Your program will be deployed on the nodes list.
   `./sd addnode --cog <cogname> --node <nodename>`
4. You can view your added nodes list by:
   `./sd list`
5. Deploy your project using:
   `./sd deploy --verbose`
   You can also use `./sd start|stop` etc. for starting your service.

ScriptRoute

A facility for distributed Internet debugging and measurement

Scriptroute is a flexible network measurement and debugging system. The tool was developed by University of Washington. It's written in Ruby - a general purpose Object Oriented scripting language.

Scriptroute's home address:
http://www.cs.washington.edu/research/networking/scriptroute/

Installing ScriptRoute on PlanetLab slice:

```
wget -nc http://www.scriptroute.org/planetlab/planetlab-
install &&\ sh ./planetlab-install
```

The installation includes scriptRoute's tools and additional useful tools.
List of tools:

ScriptRoute tools

(sr-* : type sr- + tab to get the list)

1) `sr-ally` : compare 2 hostnames/addresses to find if they are the same server
2) `sr-ping` : simple ping
3) `sr-sprobe`: an estimate for bandwidth to a host
4) `sr-traceroute`: a simple traceroute utility
5) `sr-listservers.rb`: list ScriptRoute serverlist
6) `sr-pingp` : poisson ping
7) `sr-rockettrace`: traceRoute like tool for network mapping
8) `sr-tcptraceroute`: a traceRoute utility - tcp level

Other tools installed

`ELFutils`: (eu-*) A collection of utilities to handle compiled objects.
`xmlsoft`: (xml*) Library providing XML and HTML support. Reading, Writing, parsing etc.
Includes: xmlcatalog, xmllint, xmlwf.
`Python`: An interpreted, interactive, object-oriented programming language.
It comes with many libraries, including bindings for curl, xmllib...
`gmp`: A GNU arbitrary precision arithmetic library
`yum`: (yum*) RPM packages installer/updater
`libpcap`: packet capture library
`undns`: (undns_decode) DNS router name decoder