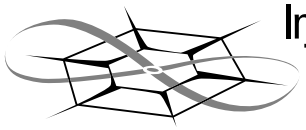
The University of Kansas



Information and Telecommunication Technology Center

Technical Report

End-to-End Monitoring of Network Traffic and Congestion Events on the Planet-Lab Network with the pmclient and pmserver Tools

Matthew Cook, Joshua Lu and Victor Frost

ITTC-FY2005-TR-30540-01

September 2004

Project Sponsor:
National Science Foundation

Abstract

The pmclient and pmserver programs are used to measure the network traffic conditions between two given nodes on the Planet-Lab network. By running the pmclient program on a Planet-Lab node, and the pmserver program on a corresponding node, the two programs produce a dynamic flow of traffic in the form of UDP packets and optionally IPv4 ICMP echo packets between the two nodes as a series of variable rate probes.

The pmclient and pmserver programs record and statistically analyze various measurements on this traffic including round trip time and time to live of UDP and IPv4 ICMP echo packets that make up the traffic. As delays in traffic increase beyond a statistically significant threshold, which is dynamically determined, or if network events such as route changes or packet loss are detected, the probing rate of the pmclient program increases to capture as much information as possible about the potential network traffic event. When traffic measurements again fall below a statistically significant threshold, the probing rate then drops to a lower level.

In addition to UDP probing delays and optional IPv4 ICMP probing delays, traceroute measurements can also be performed and recorded on a dynamic basis based on traffic conditions. The pmclient and pmserver programs record all of these measurements so that later analysis can be performed to characterize specific network traffic events.

Along with the pmclient and pmserver programs is provided a network of scripts that can be used to maintain the pmclient and pmserver programs over a large network of Planet-Lab nodes with relative ease.

Table of Contents

Table of Figures

1 – Introduction

Two methods that are typically used to monitor traffic levels and performance on a network are active and passive measurements. Passive measurements involve the observation of network traffic as it passes by the interface of a host on that network. This form of monitoring is usually implemented as a Sniffer, OCxMon, or built into devices such as routers or end node hosts. Active measurements, on the hand create artificial traffic by injecting packets into the network and measuring aspects of their transmission. Both approaches have their respective advantages and disadvantages and are more suitable in different situations and for different goals.

The goals of the National Science Foundation project *Quantifying the Temporal Characteristics of Congestion Events in the Internet* are to detect and characterize network events that have a significant impact on the end user. To further this goal, the PingMonitor suite of programs was developed to perform active measurements in order to determine the end-to-end performance of traffic generated by a generic end user application on a large network such as the Internet.

The PingMonitor programs apply an adaptive algorithm that increases and decreases the probing rate of the injected traffic based on the delays and packet losses. This system allows the PingMonitor program to minimize the amount of artificial traffic introduced to the network while at the same time collecting as much data as possible on each potential network traffic event as it occurs.

This document describes the design of the PingMonitor suite of programs, and how they are used. This included: descriptions of the testing environment in which the programs run; how this environment is set up; how to install and run the PingMonitor programs; how to set up a local environment to maintain the programs; and how to maintain and collect data from the running programs remotely.

2 – Planet-Lab

From the Planet-Lab web site:

> "PlanetLab is an open, globally distributed platform for developing, deploying and accessing planetary-scale network services. PlanetLab nodes support both short-term experiments and long-running network services…"
>
> "PlanetLab creates a unique environment in which to conduct experiments at Internet Scale. The most obvious is that network services deployed on PlanetLab experience all of the behaviors of the real Internet where the only thing predictable is unpredictability (latency, bandwidth, paths taken). A second advantage is that PlanetLab provides a diverse perspective on the Internet in terms of connection properties, network presence, and geographical location. The broad perspective on the Internet enables development and deployment of a new class of services that see the network from many different vantage points."[1]

For the purposes of the pmclient and pmserver programs, the Planet-Lab network provides a pool of geographically diverse networked Internet nodes on which network traffic measurements can be made under real world traffic conditions, but in a

homogeneous and controlled experimental environment. Currently, the National Science Foundation project *Quantifying the Temporal Characteristics of Congestion Events in the Internet* for which the pmclient and pmserver programs were developed is conducting experiments using the pmclient and pmserver programs on 42 Planet-Lab nodes located around the world.

2.1 – Node Configuration

Each Planet-Lab node is set up to allow many user accounts, or "slices" in Planet-Lab terms to access and use each node at any given time. To provide a consistent environment for each slice, each node is run as a set of virtual Linux servers, or vservers, each based on a Planet-Lab variant of RedHat Linux version 9. Each slice is provided with a bare-bones system over which users of that slice have limited superuser privileges. Each slice is assigned to a research group, and that research group can then allocate accounts on any number of Planet-Lab nodes from which they can then run their respective experiments.

Each Planet-Lab node shares a single network interface, and so to prevent separate slices running on the same physical machine from interfering with each other's network traffic, network programs running on Planet-Lab must be written using safe raw-sockets. Safe raw-sockets are an implementation of the standard BSD UNIX sockets API. For details on safe raw-sockets, please see [2]. The pmclient and pmserver programs both utilize the safe raw-sockets API.

2.2 – Current Node Allotment

The NSF project *Quantifying the Temporal Characteristics of Congestion Events in the Internet* has been allotted the "kansas_1" slice with which to conduct its experiments. The kansas_1 slice is currently running active measurements using the pmclient and pmserver programs on the Planet-Lab nodes listed in table 2.1:

Table 2.1 – Production Nodes

pli1-pa-2.hpl.hp.com	planet2.berkeley.intel-research.net
pli1-pa-3.hpl.hp.com	planetlab1.gti-dsl.nodes.planet-lab.org
planetlab5.nbgisp.com	planetlab1.comet.columbia.edu
planetlab1.cs.duke.edu	planetlab2.comet.columbia.edu
planetlab1.iis.sinica.edu.tw	planet1.leixlip.nodes.planet-lab.org
planetlab2.cs.duke.edu	planetlab01.ethz.ch
planetlab1.frankfurt.interxion.planet-lab.org	planetlab02.ethz.ch
planetlab2.frankfurt.interxion.planet-lab.org	planetlab1.diku.dk
planetlab2.iis.sinica.edu.tw	planetlab2.diku.dk
planetlab-1.stva.nodes.planet-lab.org	planet2.leixlip.nodes.planet-lab.org
planetlab-1.scla.nodes.planet-lab.org	planetlab1.koganei.wide.ad.jp
planetlab-2.scla.nodes.planet-lab.org	pl1.6test.edu.cn
planet1.seattle.intel-research.net	pl2.6test.edu.cn
planet2.seattle.intel-research.net	planet1.cavite.nodes.planet-lab.org
planetlab-2.stva.nodes.planet-lab.org	planet2.cavite.nodes.planet-lab.org
planet1.berkeley.intel-research.net	planetlab2.koganei.wide.ad.jp
planetlab1.cambridge.intel-research.net	planetlab1.cs.cornell.edu
planetlab2.cambridge.intel-research.net	planetlab1.cslab.ece.ntua.gr
planet1.pittsburgh.intel-research.net	planetlab2.cslab.ece.ntua.gr
planet2.pittsburgh.intel-research.net	planetlab1.eurecom.fr

This list was taken from the ~/nsfqos/data_retrevial/planetlab_nodes_list.txt file which is part of the pingmonitor.tar.gz archive. For more information on this file, see Setting Up the Local Environment later in this document.

In addition, the kansas_1 slice has accounts on the following nodes, which are used to test beta versions and future releases of the pmclient and pmserver programs before placing them on the main nodes. These nodes are listed in Table 2.2.

Table 2.2 – Testing Nodes

planlab1.cs.caltech.edu	kupl1.ittc.ku.edu
planetlab1.csail.mit.edu	kupl2.ittc.ku.edu

Because these nodes are used for testing, they have been outfitted with a wider variety of programs and tools than the standard production nodes. For more information on installing additional programs, see the Installing Additional Components section below.

2.3 – Installing Programs and Setting Up Node Environment

The pmclient and pmserver programs require a specific environment to be in place on any Planet-Lab node on which they run. This environment consists of a set of scripts and helper programs and a specific directory hierarchy to hold them and output from the programs. Setting up this structure is easily done using one of the remotesync.sh Bash shell scripts found in the ~/nsfqos/template directory unpacked from

the pingmonitor.tar.gz file. For more information on this file see the Setting Up The Local Environment section later in this document.

This bash script is used to upload or update the file structure of the home directory of the kansas_1 slice account on all of the 40 production Planet-Lab nodes used by kansas_1, or to upload or update files to any list of provided nodes. The program utilizes the rsync command to transfer only the files that are necessary, and will reproduce the contents of the ~/nsfqos/template directory and its subdirectories in the /home/kansas_1 folder on each node it processes.

In turn, the ~/nsfqos/template directory is a specially constructed directory of symlinks pointing to the appropriate files, folders, and scripts elsewhere in the ~/nsfqos directory that need to be uploaded. In this way, as changes are made to files and programs in anywhere in ~/nsfqos, those changes will be reflected in the template directory automatically, and can be uploaded to nodes with one of the remotesync.sh scripts. For more information about the template directory or the remotesync.sh scripts, see the readme.rtf file in the ~/nsfqos/template directory.

To run one of the remotesync.sh scripts to update all of the kansas_1 production nodes, issue the following command:

```
% ./remotesync.sh
--- foo.bar.planet-lab.org ---
Enter passphrase for key '/users/mcook/.ssh/identity':
building file list ...
36 files to consider
<output omitted>
--- foo.bar.planet-lab.org ---
Enter passphrase for key '/users/mcook/.ssh/identity':
building file list ...
36 files to consider
<output omitted>
<etc...>
```

This will run through each node, one by one, and upload the necessary files. As the script runs, each node will require the user to enter the secure shell passphrase before files can be uploaded. Because this can be tiresome, it is often wise to run this script under ssh-agent (see the ssh-agent heading of the Process Maintenance and Data Collection section later in this document).

If you wish to only update a specific node, or a list of specific nodes, simply enter the fully qualified domain names of each server after the remotesync.sh command separated by spaces on the command line.

2.4 – Directory Structure

Once setup, the Planet-Lab node will contain a set of directories placed in the home directory for the kansas_1 user /home/kansas_1/ or generally ~/kansas_1. Table 2.3 is a short description of each directory.

Table 2.3 – Directory Structure

~/backup ~/backup/client ~/backup/server	This directory contains backed up copies of program output files. The files are transferred from the ~/completed/client and ~/completed/server directories by the exec_pm_dl process maintenance script (see Process Maintenance later in this document.)
~/bin	This directory contains the pmclient and pmserver binary executable, as well as several other binary programs used by pmclient.
~/bin/scripts	This directory contains bash scripts used for process maintenance and for starting and stopping the pmclient and pmserver programs.
~/bin/source	This directory contains copies of the source code files for pmclient and pmserver, as well as backup copies of the binary executables.
~/client	This directory contains output files produced by the pmclient program.
~/server	This directory contains output files produced by the pmserver program.
~/completed/client ~/completed/server	This directory contains completed output files from the pmclient and pmserver programs respectively. The files are transferred here by each program from the ~/client and ~/server programs at the end of each day.
~/configure	This is the configuration file that is read by the pmclient program.
~/current	This is a file containing the revision number of the currently running program. This number corresponds to the revision numberd folders found in the ~/nsfqos/source/legacy source/ directory, and is currently only found on testing nodes as a convenience.

2.5 – Installing Additional Components

The remotesync program will provide the Planet-Lab node with everything that is required to run the pmclient and pmserver programs successfully. However, because each node is setup as a bare-bones system, it is often useful, especially on testing nodes, to install additional programs. This can most efficiently be done by installing the yum package manager, which has been converted for use on Planet-Lab. For more information about yum, and for instruction on installing it, please see the Installing RPM's section of the Planet-Lab User's Guide [3].

3 – The pmclient and pmserver Programs

The pmclient and pmserver programs perform active measurements on the Planet-Lab network. These two programs together comprise a complete system, and are divided into two modules according to the standard client-server model. The pmserver program simply sits on a node and listens for packets sent by the pmclient program. When it receives packets, the pmserver program quickly echoes the packets back to the client, and then records information about the packets that were received.

The pmclient program on the other hand is the workhorse of the pair. It utilizes an adaptive algorithm to continuously send UDP packets at variable rates to a remote host running the pmserver program. The client records: what packets were sent and when; when and if those packets returned from the server; and information about their trip such as round trip time, time to live, and ICMP error messages received from their transmission. In addition, the pmclient program can be configured to send IPv4 ICMP echo packets in parallel with the UDP packets, and to take periodic traceroute measurements.

3.1 – Setting Up the Local Environment

Aside from the specific environment the pmclient and pmserver programs require on each node to run, there is also a specific environment that is required by the supporting scripts and programs that help to maintain the pmclient and pmserver programs on a local system. Unpacking the pingmonitor.tar.gz file provides this environment.

Once unpacked, this file will produce a folder named nsfqos that contains a system of subfolders, each of which contains various scripts, files, and programs including the pmclient and pmserver programs and their source code. This folder should be placed in the user's home directory, and will be referenced in this document as the ~/nsfqos folder. For more information on the scripts, programs, and files in this directory, see the various readme.rtf files found throughout the folder system, and the Process Maintenance and Data Collection section later in this document.

3.2 – Program Design

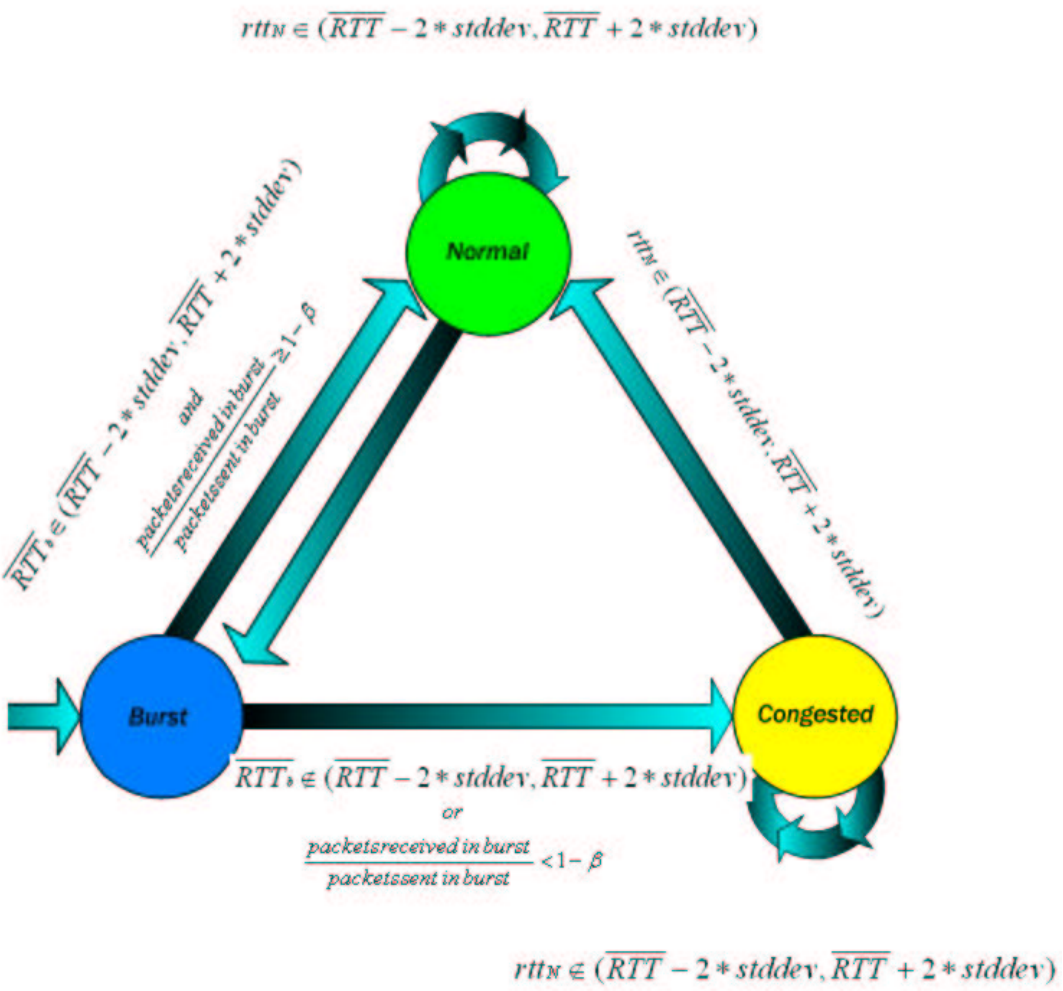
In performing measurements in a dynamic environment such as the Internet, we wish to change and adapt our measurement rate as the network transitions between "broken" and "normal" states. While the term "broken" may be an overly simplified characterization of the actual state of the network, an aspect fairly consistent among all network events that cause performance degradations are increased delays and packet losses. In order to catch these network events as early as possible and record the behavior of the network throughout the duration of the particular event, we increase our probing rate when delays reach and exceed a certain threshold or when a certain loss rate is met. Similarly as the network transitions back into a "normal" state where delays and loss rate are below certain thresholds, the probing rate is decreased.

Table 3.1 – Definition of Terms

Term	Definition
n	Burst size. Default = 30
t_α	Maximum time between bursts (s). Default = 900
t_b	Time between probes within a burst (s/probe). Default = 1
t_c	Time between probes during congestion (s/probe). Default = 5
t_n	Time between probes under normal network conditions (s/probe), Default = 30
t_l	Packet timeout (s). Default = 2
N	Total packets sent
rtt_i	Round trip time of packet containing sequence number i
β	burst loss ratio threshold. Default = 10%
$\overline{RTT}$	Mean round trip time over entire sample where $\overline{RTT} = \frac{\sum_{i=1}^{N} rtt_i}{N}$
$\overline{RTT_b}$	Mean round trip time of the most recently completed burst,

As shown in Figure 3.1, the program will be in one of three states at any given time: normal, burst, and congested. The states normal and congested correspond to the network states normal and broken as described previously where the intervals between probes are t_n and t_c respectively. While in the burst state, the program sends a burst of n probes with an interval of t_b between probes. Movement between states is dependent on the delays and packet loss rates.

Figure 3.1 – PingMonitor State Transitions



3.2.1 – Burst State

The burst state is an intermediate state when moving from the normal to congested states. It is also the initial state at the program start. A burst of n probes with an interval of t_b between probes is sent from the source node to the destination node. If the program is set to take additional traceroute measurements, then such measurements are taken at the beginning of this burst period unless the program has just started. After sending all n probes and waiting t_l seconds, if the last probe failed to return, the following are calculated:

1. mean round trip time of the probes sent during the burst: $\overline{RTT_b}$

$$\overline{RTT_b} = \frac{\sum_{j=a}^{a+n} rtt_j}{n}$$

where a is the sequence number of the first packet sent in the burst

2. packet loss ratio

$$\text{packet loss ratio} = \frac{packets\ received\ in\ burst}{packets\ sent\ in\ burst}$$

Conditions for moving to the congested state include:

1. the mean RTT of the burst set is not within 2 * the standard deviation of the mean RTT of the entire program run
2. $\overline{RTT_b} \notin (\overline{RTT} - 2 * stddev, \overline{RTT} + 2 * stddev)$
3. the packet loss ratio is greater than the packet loss threshold, β

If none of the previous conditions are met, the state is set to the normal state where the next probe will be sent in t_n seconds. Otherwise, the next probe is sent in t_c seconds and next state set to congested.

3.2.2 – Normal State

The program should stay in the normal while queuing delays are nominal and packet losses nonexistent. Probes are sent at a rate of one probe every t_n seconds so long as the RTTs are within 2 * the standard deviation of the mean of the RTT of the entire program run and the probes are not lost. However if

$rtt_N \notin (\overline{RTT} - 2 * stddev, \overline{RTT} + 2 * stddev)$, $rtt_N = \infty$ if the packet is lost

then the next state is set to burst and a burst performed immediately.

3.2.3 – Congested State

As traffic increases and queues build up in the intermediate routers between the source and destination, the RTTs will increase and be dominated by queuing delays. The program will only remain in the congested state for events that last a minimum of $t_b * n + C$ seconds. The term $t_b * n$ takes into account the time required to perform a burst while C denotes the time required for the additional measurements. The additional measurements can take a variable amount of time when transitioning from the normal state.

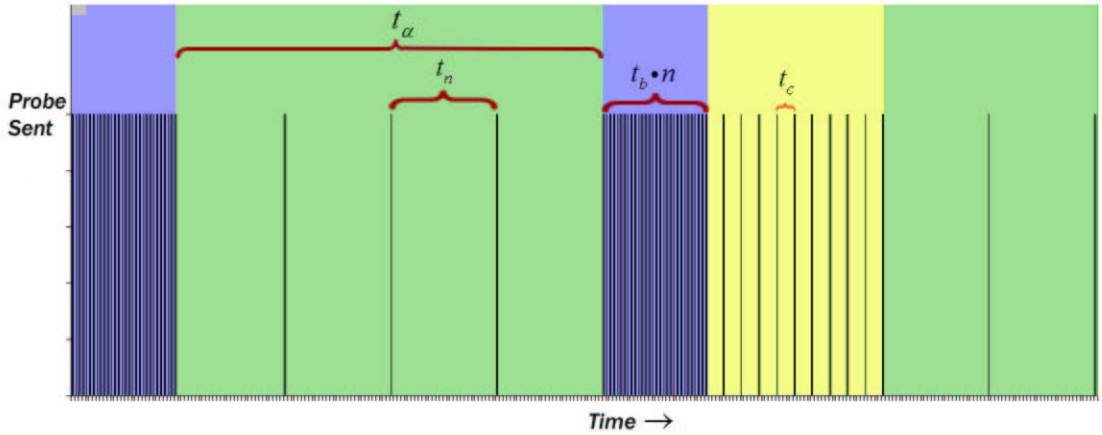
The program will remain in the congested state and continually send probes at a rate of one probe every t_c seconds while the RTTs remain outside of 2 * the standard deviation of the mean RTT of the entire program run. Once the queuing subsides and

delays return to normal, the program will set the next state to normal and schedule the next probe to be sent in t_n seconds.

An example short interval of the program run is displayed in Figure 3.2. The program performs five state transitions in the following order:

1. burst
2. normal
3. burst
4. congested
5. normal

Figure 3.2 – Short Program Run Interval



These transitions should correspond with the network displaying normal activity, becoming "broken" for period, and finally returning back to normal conditions.

3.3 – Program Specification

Both the pmclient and pmserver programs have a complement of command line arguments allowing the user to control the behavior of the programs and the algorithms used. In addition, each program produces a number of output files. These options and output files are outlined below.

3.3.1 – Command Line Options

pmclient [-igtd] [-n burst_size] [-a burst_interval] [-b burst_probe_interval] [-c congestion_probe_interval] [-n normal_probe_interval] [-l packet_timeout] [-p local_port] [-r remote_port] destination_hostname

-t	Perform traceroutes
-i	Send IPv4 ICMP echo packets in conjunction with UDP traffic.
-d	Print debugging output to standard out.
-g	Read from previous sessions on program startup and write to on program termination the mean and variance from the file, rttlog.
-n burst_size	Specify the number of probes to be sent in a burst. Default = 30
-a burst_interval	Wait a maximum of burst_interval seconds after a burst has completed before initiating another burst. Default = 900
-b burst_probe_interval	Wait burst_probe_interval seconds between sending packets during a burst. Default = 1
-c congestion_probe_interval	Wait congestion_probe_interval seconds between sending packets during congestion. Default = 5
-n normal_probe_interval	Wait normal_probe_interval seconds between sending packets during normal network conditions. Default = 30
-l packet_timeout	After a packet has been sent, specify the timeout in seconds before assuming the packet was lost. Default = 2
-p local_port	Specify the local port. Default = 8762
-r remote_port	Specify the destination port. Default = 8761
destination_hostname	Specify the hostname of the destination. If omitted, the destination is read from the configure file.

pmserver [-p port]

-p port	Specify the local port. Default = 8761

3.3.2 – Program Output

Both the pmclient and pmserver programs produce output in the form of flat ASCII UNIX text files. These files follow a regular, tabular format, and include formatted data as described in Table 3.2

Table 3.2 – Output Format Key

Tag	Format
<date&time>	M/D/YYYY H:M:S
<seq num>	Positive integers greater than 0
<rtt>	Round Trip Time measured in milliseconds to one decimal place of precision.
<route_change_flag>	This flag is deprecated, and will always appear as 0.
<stddev>	Standard Deviation measured in milliseconds.
<ttl>	Time To Live, integer values between 1 and 255.
<ICMP error type>	An integer value between 0 and 40 (see appendix A)
<ICMP error code>	An integer value between 0 and 14 (see appendix A)
<ping output>	A line of output produced by the ping program. (see the Ping Sent Log section below)

3.3.2.1 – pmclient Output

Information on packets sent, received, and potentially lost is outputted to file and stored in the ~/client bin at each PlanetLab node. All client output files contain a two-line header of the form:

```
SOURCE:, plab1.nec-labs.com, 138.15.10.55
TARGET:, planetlab1.nbgisp.com, 198.78.49.52
```

SOURCE indicates the hostname and IP address of the machine the client is running on while TARGET specifies the hostname and IP address of the machine the server is running on.

3.3.2.1.1 – Sent Packet Log: MMDDYY_sent.txt

Two columns display the time each packet was sent alongside its sequence number.

```
<date&time>, <seq num>
```

3.3.2.1.2 – Received Packet Log: MMDDYY_rtt.txt

Information about each individual packet received by the client.

```
<time&date>, <seq num>, <rtt>, <route_change_flag>, <stddev>, <ttl>
```

3.3.2.1.3 – ICMP Error Messages: MMDDYY_loss.txt

ICMP error messages received by the client. See Appendix A for a full list of message types and codes.

```
<time&date>,< ICMP  error type>,< ICMP  error code>,< sending IP address >
```

3.3.2.1.4 – Traceroute Data: MMDDYY_trace_full.txt

Raw traceroute data is stored to file if traceroutes are enabled with the –t command line option. This file will also include any errors produced by the traceroute program

3.3.2.1.5 – Ping Log: MMDDYY_ping.txt

A log of ping packets sent and received. The <ping output> tag contains a line produced by the ping program. This line contains the size of the data sent, the IP address of server, the sequence number of the ICMP message (always 1), the time to live measured as 64 minus the number of routers on the path back to the client from the server, and the round trip time in milliseconds. For more information on the ping program, see the UNIX manual page for ping.

```
<seq num>, <ping output>
```

3.3.2.1.6 – Program History: rttlog

The information necessary to calculate the mean and variance of a previous session can be stored in the file, rttlog. If invoked with the –g option, the program will check the target hostname and IP address of the local machine and compare them with the hostname and IP in the rttlog. The mean and variance are then read from file if a match is found. When the program exits, the mean and variance will then be written to the rttlog. Figure 3.3 described the format.

Figure 4 - rttlog Format

target hostname
target IP address
sum of all rtt times
sum of all rtt^2 *times*
total packets sent
total packets received

3.3.2.2 – pmserver Output

Information on packets received and potentially lost in the forward route is outputted to file and stored in the ~/server bin at each PlanetLab node. All server output files contain a single line header that contains the hostname and IP address of the machine the server is running on. For example:

```
SERVER: planetlab1.iis.sinica.edu.tw 140.109.17.180
```

3.3.2.2.1 – Packet Receive Log: MMDDYY_server.txt

Two columns, the sequence number and ttl of packets arriving at the server.

```
<seq num>, <ttl>
```

3.3.2.2.2 – Lost Packet Log: MMDDYY_forward_losses.txt

A single column contains the sequence numbers of packets that have been potentially lost in the forward direction from client to server.

```
<seq num>
```

3.4 – Running the Programs

The pmclient and pmserver programs are designed to run as background daemon processes on a Planet-Lab node. When the pmclient or pmserver program is started, it will print a short message and then return the user to the shell and run in the background. To stop the pmclient or pmserver programs once they have been started, it is necessary to send a TERM signal to the active process using the UNIX kill command.

Although it is possible to start and stop the pmclient and pmserver programs by calling the command with the correct arguments and then calling the kill command, to simplify this process, a set of start and stop scripts are installed in the ~/bin/scripts/ directory on each Planet-Lab node by the remotesync.sh install program. This method is useful to start a single instance of the program on a single Planet-Lab node, see Process Maintenance and Data Collection below for a method to maintain a system of nodes.

3.4.1 – Starting the Programs

The startpmc.sh and startpms.sh scripts are used to start the pmclient and pmserver programs respectively if they are not already running. If the programs are already running, the scripts will simply notify the user of this fact and exit. To start the pmclient program, simply run the ~/bin/scripts/startpmc.sh script like so:

```
[kansas_1@kupl1 kansas_1]$ ./bin/scripts/startpmc.sh
-----------------------------------
kupl1.ittc.ku.edu
pmclient: starting ***
usage: pmclient [options] remote host
/home/kansas_1/bin/traceroute -n 129.237.123.251 2>&1
/home/kansas_1/bin/ping -c 1 -n -w 5 129.237.123.251 | grep from
14
04
SOURCE = kupl1.ittc.ku.edu
TARGET = kupl2.ittc.ku.edu
-----------------------------------
[kansas_1@kupl1 kansas_1]$
```

Similarly, to start the pmserver process, simply run the ~/bin/scripts/startpms.sh script like so:

```
[kansas_1@kupl1 kansas_1]$ ./bin/scripts/startpms.sh
-----------------------------------
kupl1.ittc.ku.edu
pmserver: starting ***
-----------------------------------
[kansas_1@kupl1 kansas_1]$
```

3.4.2 – Stopping the Programs

The stoppmc.sh and stoppms.sh scripts are used to stop the pmclient and pmserver programs respectively if they are running. If the programs are not running, the scripts will simply notify the user of this fact and exit. To stop the pmclient program, simply run the ~/bin/scripts/stoppmc.sh script like so:

```
[kansas_1@kupl1 kansas_1]$ ./bin/scripts/stoppmc.sh
-----------------------------------
kupl1.ittc.ku.edu
27411
pmclient: stopped
ping: not running ***
-----------------------------------
[kansas_1@kupl1 kansas_1]$
```

Similarly, to stop the pmserver program, run the ~/bin/scripts/stoppms.sh script like so:

```
[kansas_1@kupl1 kansas_1]$ ./bin/scripts/stoppms.sh
-----------------------------------
kupl1.ittc.ku.edu
30264
pmserver: stopped
-----------------------------------
[kansas_1@kupl1 kansas_1]$
```

3.5 – Program Notes

There are a few scenarios under which the pmclient and pmserver programs may produce unexpected output or perform in an unexpected manor. Some of these scenarios are considered bugs and will hopefully be fixed in later releases while others are simply considered expected behavior for the program in question.

- The pmserver program creates its data files once when the program is first executed. After this, the program checks each time a packet is received if it needs to rotate its log files. Because of this design, if the pmserver program receives no packets for an extended period of time, then no new output files will be created for the duration of that time. This does not indicate that the program has crashed, but only that it has not received packets from the pmclient program.

- When a packet is received by the pmserver program with a sequence number larger than the expected sequence number, all sequence numbers between the last successfully received number and the number just received are placed on a queue in hope that the missing packets will arrive in the near future. This queue holds 5 sequence numbers, and when it is full, sequence numbers are written out of the queue into the MMDDYY_forward_losses.txt file on a first-in-last-out basis. If no sequence numbers are lost for a period of time, then a lost sequence number can remain in the queue for some time, causing it to be written to the losses file at a time much later than it was sent from the pmclient program.

- When running the pmclient program with the *–i* option to include parallel IPv4 ICMP echo packets with the UDP packets that are normally sent, in some

situations the program can become stuck. This occurs when the remote system to which pings are being directed filters ICMP echo traffic. In this situation, the call to the ping program in pmclient can hang, stopping all traffic, UDP or ICMP, from the program.

4 – Process Maintenance and Data Collection

Although it is possible to use the methods described in the previous section to start and stop the program on a small number of nodes, this process can quickly become tedious with more than just a few nodes running the programs. To make it easer to start, stop and maintain the pmclient and pmserver programs on a long list of Planet-Lab nodes, as well as collect the output files from all of these systems to a central location, a set of process maintenance and data retrieval scripts have been created and are located in the ~/nsfqos/data_retrieval folder unpacked from the pingmonitor.tar.gz file. For more information on files in this directory, and for more detailed usage information on the scripts described below, please see the ~/nsfqos/data_retrevial/readme.rtf file.

4.1 – System Requirements

These scripts utilize a number of UNIX programs that must be installed before the scripts can run. These tools are outlined below.

4.1.1 – pssh

The pssh set of programs includes parallel versions of openssh tools including ssh, scp, and rsync which allow a user to run remote procedure calls or transfer files from multiple hosts simultaneously over a secure ssh connection. By default, up to 32 ssh processes can be run at a given time allowing program updates to be deployed, data sets retrieved, and remote scripts to be run on a large set of nodes all at once.

The pssh program is used by the process maintenance scripts to remotely run the startpmc.sh, startpms.sh, stoppmc.sh, and stoppms.sh start and stop scripts described in the Running the Programs section on each Planet-Lab node. A RedHat Package Manager file containing the pssh programs is located in the ~/nsfqos/tools directory, or can be obtained from [4].

4.1.2 – ssh-agent

Because these scripts utilize the secure shell as a transport medium for moving files and running commands on each node, this process required the user to enter a passphrase for each connection that is made. The ssh-agent program allows a user to enter his or her ssh-passphrase once and then have the ssh-agent program take care of authentication with each subsequent connection to a remote host.

To use the ssh-agent program, the Planet-Lab ssh keys found in the ~/nsfqos/ssh_keys directory must first be placed into the current user's ~/.ssh/ folder. Once this is done, enter the following commands.

```
$ ssh-agent bash
$ ssh-add
Enter passphrase for /Users/admin/.ssh/identity: <enter passphrase>
Identity added: /Users/admin/.ssh/identity (/Users/admin/.ssh/identity)
$
```

You can now connect to the Planet-Lab nodes associated with that ssh-key without entering your passphrase, or run any of the process maintenance and data retrieval scripts unattended. Note that once you log out or close the xterm window in which you started ssh-agent, the ssh-agent program will exit, and will no longer fill in

passwords for you. You must either remain logged in, or use the screen command as outlined below in the Optional Tools section. The ssh-agent program is included with most distributions as part of the secure shell package.

4.2 – Optional Tools

In addition to the required programs above, there are a few optional tools that can make running the process maintenance scripts much easer. These tools are outlined below.

4.2.1 – screen

The screen command allows you to start a terminal session, begin running programs, and then detach that terminal session from your physical terminal. That session continues to run in the background, and can later be reattached to your terminal, or to any other remote terminal you log in from. For more information about the screen command, please see the UNIX manual page for screen.

4.3 – The exec_pm_dl Script

The exec_pm_dl script is the main process maintenance and data retrieval script. It can be found in the ~/nsfqos/data_retrieval directory. The script has 3 main functions, each of which is turned on or off through the use of a command line flag. Although it is possible to run this script by hand, the exec_pm_dl script is designed to be run by the getpmdata automation script that is outlined in the next section.

When executed with the *–r* flag, the exec_pm_dl script will connect to each node listed in the ~/nsfqos/data_retrevial/planetlab_node_list.txt file and run the startpmc.sh and startpms.sh scripts on that node. This has the effect of starting the pmclient or pmserver program if it is not running, or no effect if the program is already running as outlined in the Starting the Programs section previously. The script will output which nodes it was and was not able to connect to, and the output of the start script from each node on which it was run.

When run with the *–t* flag, the exec_pm_dl script will make an rsync connection to each node (listed internally in the script), and transfer files from that node's ~/client/ and ~/server/ directories to appropriate folders in the /projects/nsfqos/fall_2004 directory locally. The script will output the information about each transfer as it takes place. The script will make 5 attempts to connect to each server to make the transfer.

When run with the *–b* flag, the exec_pm_dl script will connect to each node outlined by the ~/nsfqos/data_retrevial/planetlab_node_list.txt file and move any files it finds in the ~/completed/client/ and ~/completed/server/ folders on each node into the ~/backup/client and ~/backup/server folders respectively on the same node. The script will output the success or failure of each connection and attempt to move files.

4.4 – The getpmdata Automation Script

The getpmdata script, which can be found in the ~/nsfqos/data_retrevial/ folder, is used to call the exec_pm_dl script on a regular basis though the day. The script utilizes the at deferred command scheduler to run the exec_pm_dl script every two hours over a 24 hour period, and then set up the same schedule for the next day. This process will

continue indefinitely until the script is stopped. After each command is run, the at daemon will email the output of the command to the user that ran the script.

The getpmdata script is set to call the exec_pm_dl script with the *–t* and *–r* options every two hours to keep the processes running and transfer new data, and with the *–t*, *–r*, and *–b* options once a day around midnight to additionally backup the data from the previous day. The timings of the commands are offset from the top of the hour by 18 minutes to allow the pmclient and pmserver programs time to change to new output files at midnight. Because this script is designed to run continuously, it is a good idea to run it under ssh-agent.

4.5 – The stopall.sh Script

The stopall.sh script is used to connect to all of the Planet-Lab nodes listed in the ~/nsfqos/data_retrevial/planetlab_node_list.txt file and stop the pmclient and pmserver programs running there using the stoppmc.sh and stoppms.sh scripts located on each node and described in the Stopping the Programs section above. This script is provided as a convenience and is generally used to stop all of the processes running on the nodes before updating them with the remotesync.sh script to a new version and then restarting them.

When the script is run, it will output the success or failure of each connection and the output of the stop-scripts run on each node. If you have Planet-Lab nodes that are difficult to reach due to network congestion or distance, it is a good idea to run this script several times to ensure that all nodes have been successfully stopped.

4.6 – Examples

What follows is an annotated example of the typical process of updating the program on all of the Planet-Lab nodes. This example will illustrate stopping the local maintenance scripts, stopping pmclient and pmserver processes on all of the nodes, updating the nodes with new files, and restarting the maintenance scripts. With minor changes, this process is identical to installing and starting the programs on a set of nodes for the first time, or moving the maintenance scripts to a new local machine.

The first task is to stop the local maintenance scripts. You can do this by killing the already running getpmdata script, and then removing any remaining jobs from the atq with the atrm command.

```
% kill `/sbin/pidof getpmdata` `/sbin/pidof sleep`
% atq
733     2004-08-15 00:18 a mcook
734     2004-08-15 02:18 a mcook
735     2004-08-15 04:18 a mcook
736     2004-08-15 06:18 a mcook
737     2004-08-15 08:18 a mcook
738     2004-08-15 10:18 a mcook
740     2004-08-14 14:18 a mcook
741     2004-08-14 16:18 a mcook
742     2004-08-14 18:18 a mcook
743     2004-08-14 20:18 a mcook
744     2004-08-14 22:18 a mcook
% atrm `perl -e 'foreach(733..744) { print "$_ "; }'`
```

This small perl command will list number 733 – 744 on the command line separated by spaces after the atrm command. You could just as easily list them yourself.

Now we need to stop all of the server processes with the ~/nsfqos/data_retrevial/stopall.sh script. Because this will involve making many ssh connections to the Planet-Lab nodes, we will run this command under ssh-agent.

```
% ssh-agent bash
$ ssh-add
Enter passphrase for /users/mcook/.ssh/identity: <enter passphrase>
Identity added: /users/mcook/.ssh/identity (/users/mcook/.ssh/identity)
$ cd ~/nsfqos/data_retrieval/
$ ./stopall.sh
<output omitted>
```

As noted in the stopall.sh Script section above, it may be necessary to run this script several times to ensure that all processes on all nodes have been stopped.

Next we will update the nodes with the ~/nsfqos/template/remotesync.sh script. Note that we are still under ssh-agent.

```
$ cd ../template/
$ ./remotesync.sh
<output omitted>
$ exit
exit
%
```

It's a good idea at this point to make note of any nodes that could not be updated because of network problems. They will need to be updated at a later date when and if they become reachable again. Also note that we have exited the bash shell and so stopped the ssh-agent.

Now that the nodes are updated, all that remains is to restart the maintenance scripts again. We will do this by first creating a virtual terminal with the screen command, starting ssh-agent within this terminal, running the getpmdata script as a background process, and then detaching the screen so that it can continue running until we need to deal with it again. If the script had previously been started under the screen command, then it may have been more convenient to reconnect to that screen before starting this process, and then perform these commands all from within that terminal.

```
% screen –S “getpmdata process”
% cd ~/nsfqos/data_retrieval/
/users/mcook/nsfqos/data_retrieval
% ssh-agent bash
$ ssh-add
Enter passphrase for /users/mcook/.ssh/identity: <enter passphrase>
Identity added: /users/mcook/.ssh/identity (/users/mcook/.ssh/identity)
$ ./getpmdata &
$^a^d
```

The last command is issued by the key combination Ctrl-a Ctrl-d, and causes the screen program to detach from the currently running virtual terminal and return to the command line. Now the exec_pm_dl script will run at the next 2-hour interval and restart the pmclient and pmserver programs on each node. It is safe to close your command window and logout, however, the local system must remain on for the scripts to run.

Appendix A – ICMP Types and Codes

The following table of ICMP message types and codes is derived from John Postel's original paper on the ICMP protocol, now referenced as RFC 792. For more information on the ICMP protocol, see references [5] – [11]

```
Type    Name                                    Reference
----    ------------------------                ---------
  0     Echo Reply                               [RFC792]
        Codes
            0  No Code

  1     Unassigned                                  [JBP]
  2     Unassigned                                  [JBP]
  3     Destination Unreachable                  [RFC792]
        Codes
            0  Net Unreachable
            1  Host Unreachable
            2  Protocol Unreachable
            3  Port Unreachable
            4  Fragmentation Needed and Don't Fragment was Set
            5  Source Route Failed
            6  Destination Network Unknown
            7  Destination Host Unknown
            8  Source Host Isolated
            9  Communication with Destination Network is
               Administratively Prohibited
           10  Communication with Destination Host is
               Administratively Prohibited
           11  Destination Network Unreachable for Type of Service
           12  Destination Host Unreachable for Type of Service
           13  Communication Administratively Prohibited      [RFC1812]
           14  Host Precedence Violation                      [RFC1812]
           15  Precedence cutoff in effect                    [RFC1812]

  4     Source Quench                            [RFC792]
        Codes
            0  No Code
  5     Redirect                                 [RFC792]
        Codes
            0  Redirect Datagram for the Network (or subnet)
            1  Redirect Datagram for the Host
            2  Redirect Datagram for the Type of Service and Network
            3  Redirect Datagram for the Type of Service and Host
  6     Alternate Host Address                      [JBP]
        Codes
            0  Alternate Address for Host
  7     Unassigned                                  [JBP]
  8     Echo                                     [RFC792]
        Codes
            0  No Code
  9     Router Advertisement                    [RFC1256]
        Codes
            0  No Code
 10     Router Selection                        [RFC1256]
        Codes
            0  No Code
 11     Time Exceeded                            [RFC792]
        Codes
            0  Time to Live exceeded in Transit
            1  Fragment Reassembly Time Exceeded
 12     Parameter Problem                        [RFC792]
        Codes
            0  Pointer indicates the error
            1  Missing a Required Option        [RFC1108]
            2  Bad Length
 13     Timestamp                                [RFC792]
        Codes
            0  No Code
```

```
 14     Timestamp Reply                          [RFC792]
        Codes
            0  No Code
 15     Information Request                      [RFC792]
        Codes
            0  No Code
 16     Information Reply                        [RFC792]
        Codes
            0  No Code
 17     Address Mask Request                     [RFC950]
        Codes
            0  No Code
 18     Address Mask Reply                       [RFC950]
        Codes
            0  No Code
 19     Reserved (for Security)                    [Solo]
 20-29  Reserved (for Robustness Experiment)        [ZSu]
 30     Traceroute                              [RFC1393]
 31     Datagram Conversion Error               [RFC1475]
 32     Mobile Host Redirect               [David Johnson]
 33     IPv6 Where-Are-You                  [Bill Simpson]
 34     IPv6 I-Am-Here                      [Bill Simpson]
 35     Mobile Registration Request         [Bill Simpson]
 36     Mobile Registration Reply           [Bill Simpson]
 39     SKIP                                    [Markson]
 40     Photuris                                [Simpson]

Code
0       Reserved
1       unknown security parameters index
2       valid security parameters, but authentication failed
3       valid security parameters, but decryption failed
```

References

1. “Planet-Lab Homepage”, 2004 - http://www.planet-lab.org/

2. PlanetLab Team, “Scout Module API: Safe Raw Sockets”, 2004 - http://www.planet-lab.org/raw_sockets/api.html

3. “PlanetLab Documentation: User’s Guide”, 2004 - http://www.planet-lab.org/doc/UsersGuide.php

4. Chun, Brent N., “pssh”, November 2003 - http://www.theether.org/pssh/

5. Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, USC/Information Sciences Institute, September 1981. - http://www.faqs.org/rfcs/rfc792.html

6. Mogul, J., and J. Postel, "Internet Standard Subnetting Procedure", STD 5, RFC 950, Stanford, USC/Information Sciences Institute, August 1985. - http://www.faqs.org/rfcs/rfc950.html

7. Kent, S., "U.S. Department of Defense Security Options for the Internet Protocol", RFC 1108, November 1991. - http://www.faqs.org/rfcs/rfc1108.html

8. Deering, S., Editor, "ICMP Router Discovery Messages", RFC 1256, Xerox PARC, September 1991. - http://www.faqs.org/rfcs/rfc1256.html

9. Malkin, G., "Traceroute Using an IP Option", RFC 1393, Xylogics, Inc., January 1993. - http://www.faqs.org/rfcs/rfc1393.html

10. Ullmann, R., "TP/IX: The Next Internet", RFC 1475, Process Software Corporation, June 1993. - http://www.faqs.org/rfcs/rfc1475.html

11. Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, Cisco Systems, June 1995. - http://www.faqs.org/rfcs/rfc1812.html