



UNIVERSITY OF
SOUTH CAROLINA

**PERFSONAR
LAB SERIES**

Book Version: **01-25-2021**

Principal Investigator: Jorge Crichigno



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Lab 1: Configuring Administrative Information Using perfSONAR Toolkit GUI

Lab 2: PerfSONAR Metrics and Tools

Lab 3: Configuring Regular Tests Using perfSONAR GUI

Lab 4: Configuring Regular Tests Using pScheduler CLI Part I

Lab 5: Configuring Regular Tests Using pScheduler CLI Part II

Lab 6: Bandwidth-delay Product and TCP Buffer Size

Lab 7: Configuring Regular Tests Using a pSConfig Template

Lab 8: perfSONAR Monitoring and Debugging Dashboard

Lab 9: pSConfig Web Administrator

Lab 10: Configuring pScheduler Limits



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 1: Configuring Administrative Information Using perfSONAR Toolkit GUI

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	4
Lab roadmap.....	4
1 Introduction.....	4
2 Configuring administrative information	5
2.1 Adding a web user	5
2.2 Accessing the administrative information interface	7
2.3 Filling up administrative information.....	7
2.4 Adding node metadata	11
2.5 Adding a community	13
References.....	16

Overview

This lab provides an introduction to perfSONAR Toolkit. It shows how to configure the administrative information of a perfSONAR node using the Graphical User Interface (GUI).

Objectives

By the end of this lab, the user will:

1. Understand perfSONAR GUI.
2. Access to perfSONAR Toolkit GUI.
3. Configure the administrative information.
4. Visualize the administrative information of a perfSONAR node.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled as perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

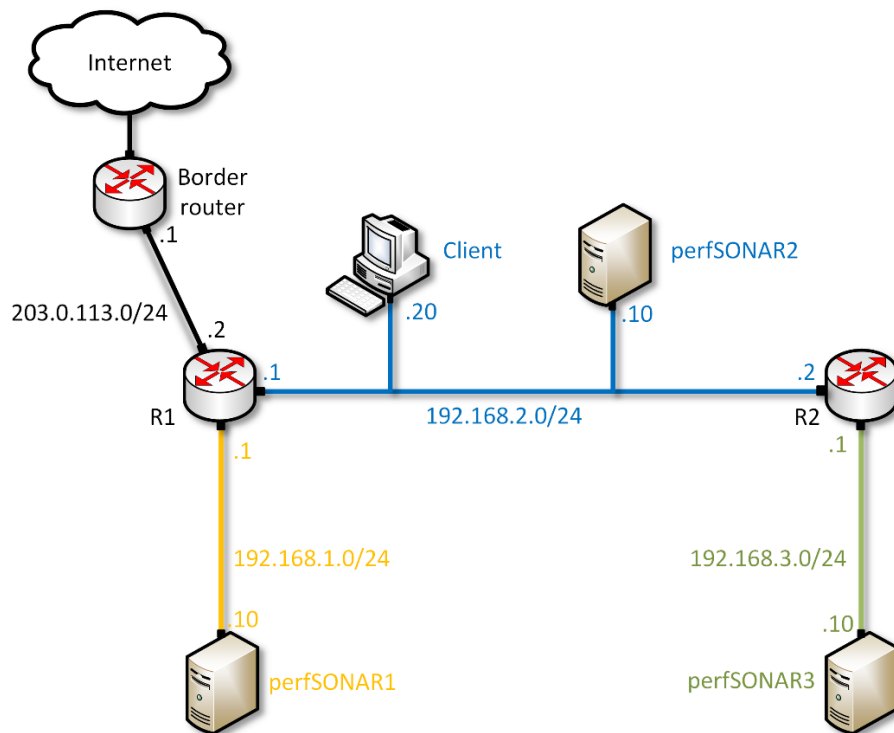


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Configuring Administrative Information.

1 Introduction

Networks are designed to support diverse mixtures of hardware and protocols, especially in large collaborations. Interoperability takes precedence in most cases, along with local control and policy being preserved. Reason that, actions taken by one organization can affect the performance of users in another organization. A global monitoring framework is required to reliably discover and mitigate these issues. Monitoring within a single domain is a common and accepted practice but cross-domain performance monitoring is difficult to do with traditional tools¹.

perfSONAR is a tool which offers web services-based infrastructure from collecting and diagnosing network performance. perfSONAR makes it possible to diagnose problems on networks quickly and easily, providing a collection of tools for performing and sharing end-to-end network measurements. perfSONAR is used to diagnose performance issues such as latency, achievable bandwidth, packet loss, and many others². While perfSONAR is currently focused on reporting network metrics, it is designed to be flexible enough to handle new metrics from technologies as middleware or monitoring³.

The perfSONAR project is currently deployed in over 1,700 locations around the world. Its main feature relies on network troubleshooting. perfSONAR has been developed through an international collaboration led by Internet2, ESnet, Indiana University, and GÉANT⁴.

The perfSONAR Toolkit Graphical User Interface (GUI) is a fully enclosed measurement infrastructure packaged as a Linux distribution. perfSONAR Toolkit GUI belongs to the visualization layer, as shown in the figure 2. In this lab, the user will configure the administrative information using perfSONAR Toolkit GUI.

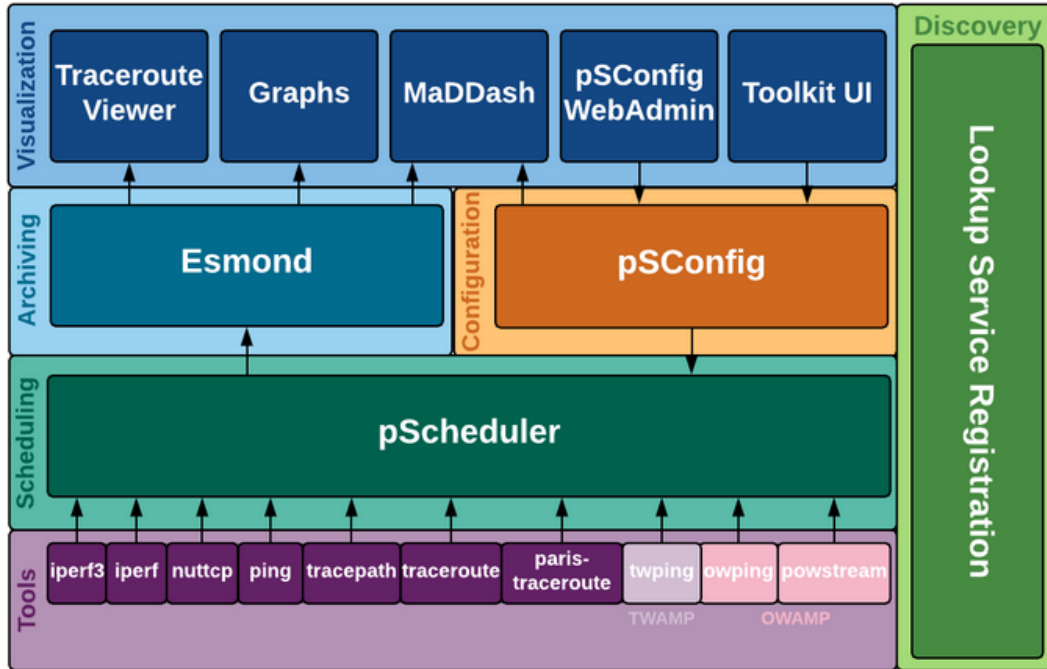


Figure 2. perfSONAR layers².

2 Configuring administrative information

The perfSONAR Toolkit GUI allows the user to enter contact and location information about a perfSONAR node. Once the perfSONAR Toolkit is installed and it is booted for the first time, the first step consists on adding a user with the privileges to edit and manage the administrative information.

2.1 Adding a web user

The perfSONAR Toolkit provides utilities for adding, deleting and modifying user's privileges to access the web interface. All of these tasks can be done through the perfSONAR command-line interface (CLI).

Step 1. In the topology, click on perfSONAR1 and enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.10.1.el7.x86_64 on an x86_64

perfsonar1 login: admin
Password:
Last login: Mon Apr 15 10:12:00 on tty1
Welcome to the perfSONAR Toolkit v4.1.6-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

    /usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar1 ~]$_
```

Step 2. In order to create a new user, type the command displayed down below. If a password is required, type `admin` as password.

```
sudo /usr/lib/perfsonar/scripts/nptoolkit-configure.py
```

```
[admin@perfsonar1 ~]$_ sudo /usr/lib/perfsonar/scripts/nptoolkit-configure.py
[sudo] password for admin:

perfSONAR Toolkit customization script

1. Change Timezone
2. Manage Web Users
0. exit

Make a selection: _
```

Step 3. Select *Manage Web Users* typing `2` and then hit *Enter* to proceed.

```
perfSONAR Toolkit customization script

1. Change Timezone
2. Manage Web Users
0. exit

Make a selection: 2_
```

Step 4. Select *Add a new user* typing `1` and then hit *Enter* to proceed

```
Welcome to the perfSONAR Toolkit user administration program.
This program will help you administer users.
You may configure any of the options below with this program:
 1. Add a new user
 2. Delete a user
 3. Change a user's password
 0. exit
Make a selection: 1
```

Step 5. Type `admin` as the username and `admin` as the password, the user will be required twice to enter the password. Notice that the password will not be displayed while typing it. In the future, the user can change the password running the same script and selecting option `3`.

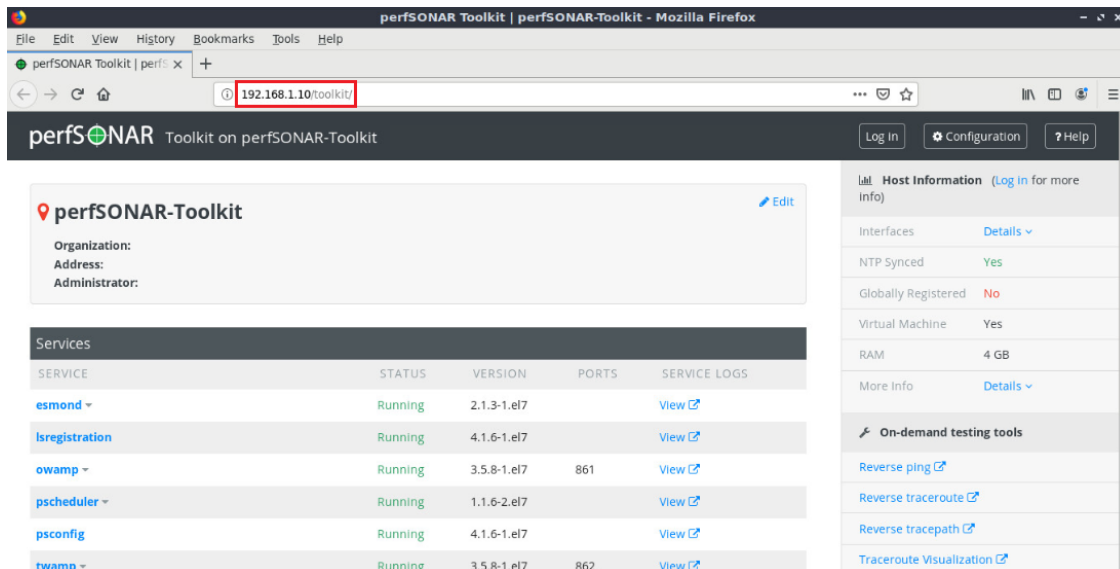

```
Enter the user whose account you'd like to add. Just hit enter to exit: admin
New password:
Re-type new password:
Adding password for user admin
```

2.2 Accessing the administrative information interface

Step 1. On the Client host, open web browser located on the desktop.



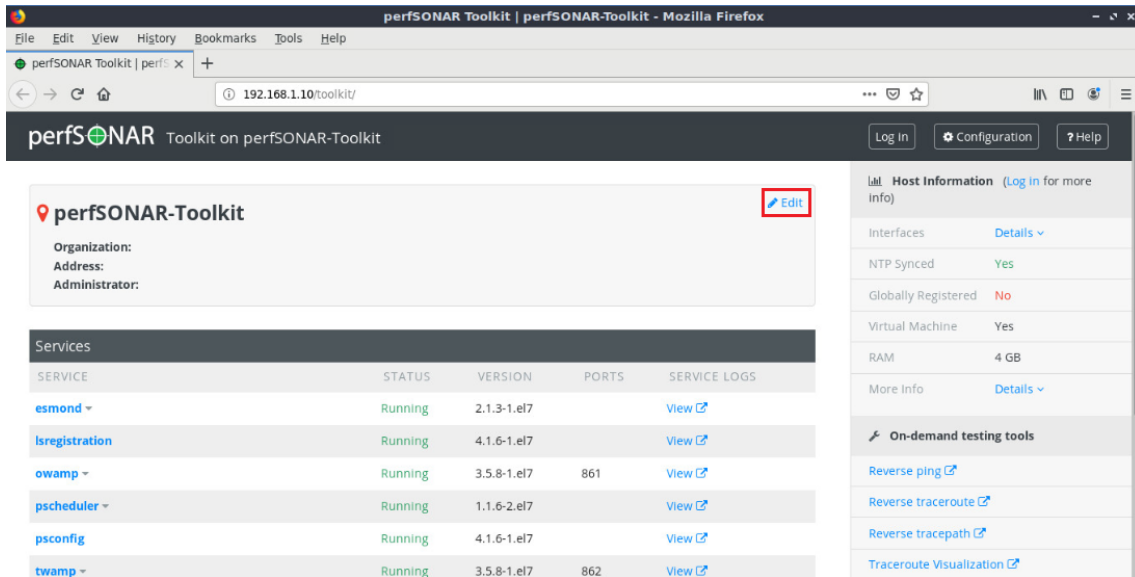
Step 2. On the address bar, type the IP address of perfSONAR1 Toolkit node *which is 192.168.1.10*. The user will see the perfSONAR Toolkit web interface.



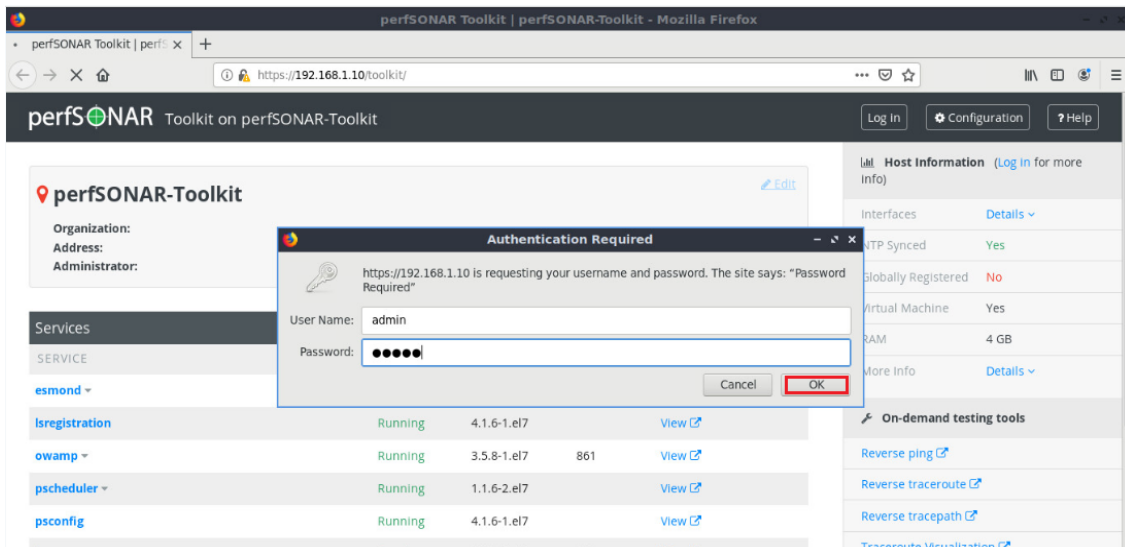
2.3 Filling up administrative information

In this section the user will fill out a form with the corresponding administrative information. Note that the information provided in this section is just for training purposes, and it is valid only for this lab. The user may change this information depending on the characteristics of the node.

Step 1. Click on edit.



Step 2. The user will be given an authentication screen. Type `admin` as the *User Name* and `admin` as the *Password* then, click on *OK*.



Step 3. In this step the user will fill a form with the administrative information. This information may change depending on the organization, administrator information, and location of the node. Fill the form with the following information:

- *Organization Name:* The name of the organization to which this host belongs.

- **Administrator Name:** The full name of a person to contact about this host.
- **Administrator email:** The email address where correspondence regarding this host may be sent. Since this e-mail address should be used only for communication related to the operations of the specific node, it is highly recommended that a role or group e-mail address is used instead of a personal one.
- **City:** The city where the host resides.
- **Country:** The country where the host resides.
- **State/Province:** The state, province or other country-specific region where the host resides. May be the 2-letter abbreviation if applicable.
- **ZIP/Postal Code:** The postal code of the location where the host resides.
- **Latitude:** The latitude of the host as a decimal number between -90 and 90. Note that if you are in the southern hemisphere, this value should be negative.
- **Longitude:** The longitude of the host as a decimal number between -180 and 180. Note that if the node is in the western hemisphere, this value should be negative.

The screenshot shows the 'Administrative Information' tab in the perfSONAR Toolkit GUI. The form contains the following fields and values:

- Organization Name: University of South Carolina
- Administrator Name: Administrator
- Administrator Email: admin@admin.com
- City: Columbia
- Country: United States
- State/Province: South Carolina
- ZIP/Postal Code: 29201
- Latitude: 34.0007104
- Longitude: -81.0348144

A privacy policy notice is displayed: "All the information you provide on this page will be sent, recorded and made publicly available on the global perfSONAR Lookup Service. For privacy reasons, we recommend you use a role or group name and related email address to be registered. Any personal information you would provide will be on your own responsibility and will by no means represent an obligation for the perfSONAR project. See our [Privacy Policy](#) for more information." Below this notice is a checkbox labeled "I agree to the perfSONAR Privacy Policy".

The 'Resources' sidebar on the right contains the following links:

- [Editing Host Information](#)
- [Managing Communities](#)
- [Privacy Policy](#)

At the bottom of the form, a dark bar displays the message: "Metadata: You've made changes that haven't been saved." with "Cancel" and "Save" buttons.

Step 4. Click on the check box in order to agree to the perfSONAR Policy.

Lab 1: Configuring Administrative Information Using perfSONAR Toolkit GUI

Administrative Information Host Tests

Organization Name
University of South Carolina

Administrator Name
Administrator

Administrator Email
admin@admin.com

City
Columbia

Country
United States

State/Province
South Carolina

ZIP/Postal Code
29201

Latitude
34.0007104

Longitude
-81.0348144

All the information you provide on this page will be sent, recorded and made publicly available on the global perfSONAR Lookup Service. For privacy reasons, we recommend you use a role or group name and related email address to be registered. Any personal information you would provide will be on your own responsibility and will by no means represent an obligation for the perfSONAR project. See our [Privacy Policy](#) for more information.

I agree to the perfSONAR Privacy Policy

Resources

- [Editing Host Information](#)
- [Managing Communities](#)
- [Privacy Policy](#)

You've made changes that haven't been saved.

Cancel Save

Step 5. Click *Save* to apply the changes.

Administrative Information Host Tests

Organization Name
University of South Carolina

Administrator Name
Administrator

Administrator Email
admin@admin.com

City
Columbia

Country
United States

State/Province
South Carolina

ZIP/Postal Code
29201

Latitude
34.0007104

Longitude
-81.0348144

All the information you provide on this page will be sent, recorded and made publicly available on the global perfSONAR Lookup Service. For privacy reasons, we recommend you use a role or group name and related email address to be registered. Any personal information you would provide will be on your own responsibility and will by no means represent an obligation for the perfSONAR project. See our [Privacy Policy](#) for more information.

I agree to the perfSONAR Privacy Policy

Resources

- [Editing Host Information](#)
- [Managing Communities](#)
- [Privacy Policy](#)

You've made changes that haven't been saved.

Cancel Save

Step 6. After applying the changes, the user will see the administrative information on the public dashboard. To see this information, click on *View public dashboard*.

The screenshot shows the 'Administrative Information' configuration page in the perfSONAR Toolkit. The page includes a header with the perfSONAR logo and navigation links for 'View public dashboard', 'Configuration', and 'Help'. Below the header, there are tabs for 'Administrative Information', 'Host', and 'Tests'. The main form contains several input fields: 'Organization Name' (University of South Carolina), 'Administrator Name' (Administrator), 'Administrator Email' (admin@admin.com), 'City' (Columbia), 'Country' (United States), 'State/Province' (South Carolina), and 'ZIP/Postal Code' (29201). A checkbox for 'I agree to the perfSONAR Privacy Policy' is checked. A 'Resources' sidebar on the right lists links for 'Editing Host Information', 'Managing Communities', and 'Privacy Policy'. At the bottom, there are 'Cancel' and 'Save' buttons.

Step 7. The user can verify the given information on the public dashboard of perfSONAR Toolkit.

The screenshot shows the public dashboard for a perfSONAR node. The header includes the perfSONAR logo and navigation links for 'Log in', 'Configuration', and 'Help'. The main content area displays the node's details: 'perfSONAR-Toolkit' with an 'Edit' link. Below this, the organization and administrator information are listed: 'Organization: University of South Carolina', 'Address: Columbia, SC 29201 US (map)', and 'Administrator: Administrator (admin@admin.com)'. A 'Services' table lists various services with their status, version, ports, and service logs. A 'Test Results' section shows '(No Results)' and a 'Configure tests' button. On the right, a 'Host Information' sidebar provides details about the node's configuration, including interfaces, NTP sync status, globally registered status, node role, access policy, virtual machine status, and RAM. It also lists 'On-demand testing tools' such as reverse ping, reverse traceroute, reverse tracepath, and traceroute visualization.

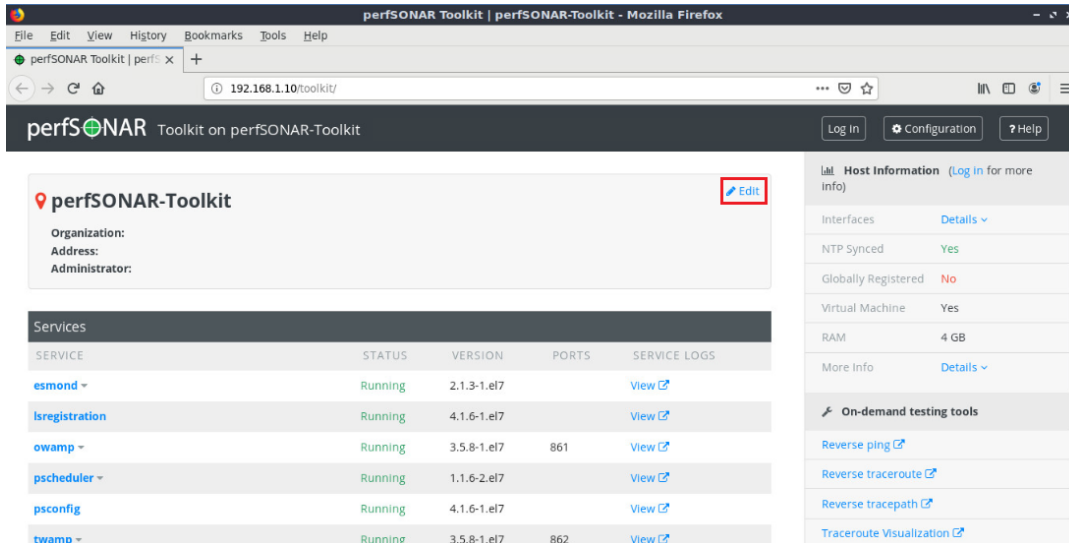
SERVICE	STATUS	VERSION	PORTS	SERVICE LOGS
esmond	Running	2.1.3-1.e17		View
lsregistration	Running	4.1-1.e17		View
owamp	Running	3.5.8-1.e17	861	View
pscheduler	Running	1.1.5-2.e17		View
psconfig	Running	4.1.5-1.e17		View
twamp	Running	3.5.8-1.e17	862	View

2.4 Adding node metadata

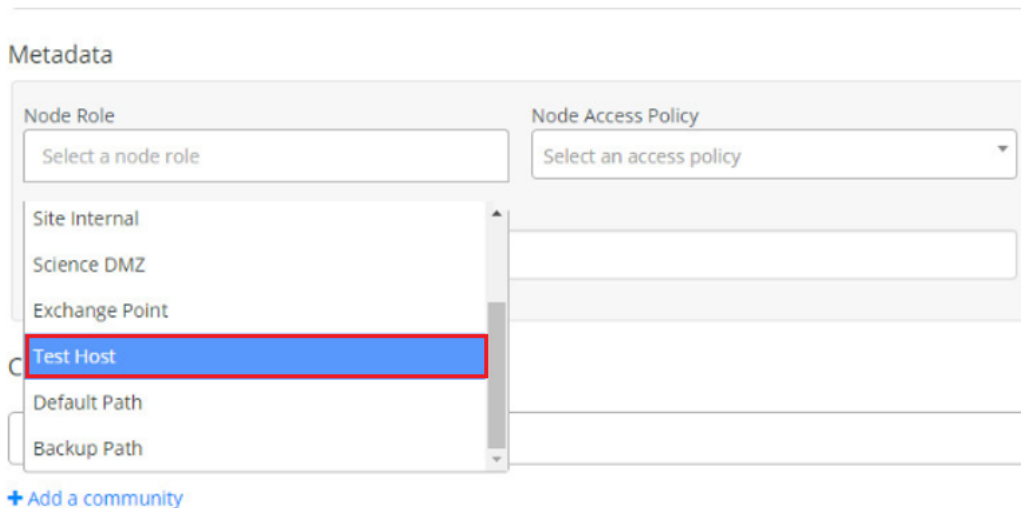
The perfSONAR project maintains a graphical interface to the services directory of all perfSONAR nodes. The node metadata are tags that can be used to describe a host in the global node directory page. There are two types of metadata tags:

- **Node Role:** It describes the node roles in the domain. It helps potential users of this node to recognize the place of node installation in the domain of the owner. The user can select multiple roles for a node.
- **Node Access Policy:** It is used to indicate the access policy for a node. These policies could be: public access node, private with no access, R&E only, or with limited access. The user can select only one Access Policy for a node.

Step 1. Click on *Edit*, if the user is required to authenticate, type `admin` as the username and `admin` as password.



Step 2. In order to add a Node Role, under Metadata, click on the field *Node Role*. A drop-down list shows with possible values. Click on *Test Host* value to select it. The user can repeat this step to add more tags.



Step 3. In order to add a Node Access Policy, under Metadata, click in the field *Node Access Policy*. A drop-down list shows with possible values. Click on a *R&E Only*.

Metadata

Node Role × Test Host ×	Node Access Policy Select an access policy ▲
Access Policy Notes <input type="text"/>	Public R&E Only Private Limited

Communities

Select communities

+ Add a community

Step 4. The user may also add a descriptive note in *Access Policy Notes* field which is a human readable text that can optionally be added to help further describe the access policy.

Metadata

Node Role × Test Host ×	Node Access Policy R&E Only × ▼
Access Policy Notes <input type="text" value="Access policy text..!"/>	

Communities

Select communities

+ Add a community

Step 5. Click *Save* to apply the changes.

2.5 Adding a community

Communities are self-defined tags that can be used as a means to search for a host on the Global node directory page. There are two ways to select from existing communities, either by selecting from the list of existing communities or by typing the known community (note that communities are case-sensitive).

Step 1. Under *Communities*, click the field *Select communities*. A list will be shown with existing communities. Select one, for example *ESNet*.

Communities

ESNet

ESNet

ESnet

Esnet

esnet

Esnet

Step 2. The user can also create a community tag clicking on *Add a Community*. An entry box will be displayed.

Communities

× ESNet

+ Add a community

Enter a community name

Add

Step 3. Write the community name. For example, type UofSC and then click on *Add*.

Communities

× ESNet

+ Add a community

UofSC

Add

Step 4. Click on the checkbox to agree with perfSONAR policy.

The screenshot shows a configuration form for administrative information. The form includes the following fields and values:

- Organization Name: University of South Carolina
- Administrator Name: Administrator
- Administrator Email: admin@admin.com
- City: Columbia
- Country: United States
- State/Province: South Carolina
- ZIP/Postal Code: 29201
- Latitude: 34.0007104
- Longitude: -81.0348144

On the right side of the form, there is a privacy notice: "All the information you provide on this page will be sent, recorded and made publicly available on the global perfSONAR Lookup Service. For privacy reasons, we recommend you use a role or group name and related email address to be registered. Any personal information you would provide will be on your own responsibility and will by no means represent an obligation for the perfSONAR project. See our [Privacy Policy](#) for more information." Below the notice is a checkbox labeled "I agree to the perfSONAR Privacy Policy" which is checked and highlighted with a red box.

Step 5. Click on *Save* to apply the changes.

The screenshot shows a community management interface. At the top, there are two tabs: "ESNet" and "UofSC". Below the tabs is a "+ Add a community" link. Underneath is a form with the label "Enter a community name" and an "Add" button. At the bottom of the interface, there is a dark grey bar with the text "You've made changes that haven't been saved." and two buttons: "Cancel" and "Save". The "Save" button is highlighted with a red box.

Step 6. Click on *View public dashboard*.

The screenshot shows the perfSONAR Toolkit public dashboard. The page title is "perfSONAR Toolkit on perfSONAR-Toolkit". In the top right corner, there are three buttons: "View public dashboard" (highlighted with a red box), "Configuration", and "Help". Below the title bar, there is a breadcrumb trail: "Home / Configuration". The main content area has three tabs: "Administrative Information" (selected), "Host", and "Tests". The "Administrative Information" tab shows the same configuration form as in Step 5, with the "I agree to the perfSONAR Privacy Policy" checkbox checked. On the right side, there is a "Resources" sidebar with three links: "Editing Host Information", "Managing Communities", and "Privacy Policy". At the bottom of the page, there is a dark grey bar with "Cancel" and "Save" buttons.

Step 7. The user can verify the given information on the public dashboard of perfSONAR Toolkit.

The screenshot displays the perfSONAR Toolkit GUI. At the top, there is a navigation bar with 'Log in', 'Configuration', and 'Help' buttons. The main content area is divided into two columns. The left column shows the 'perfSONAR-Toolkit' header with an 'Edit' link, followed by organizational details: 'Organization: University of South Carolina', 'Address: Columbia, SC 29201 US (map)', and 'Administrator: Administrator (admin@admin.com)'. Below this is a 'Services' table with columns for SERVICE, STATUS, VERSION, PORTS, and SERVICE LOGS. The table lists several services: esmond, lsregistration, owamp, pscheduler, psconfig, and twamp, all with a status of 'Running'. At the bottom of the left column is a 'Test Results' section showing '(No Results)' and a 'Configure tests' button. The right column is titled 'Host Information' and contains a list of system details: Interfaces (Details), NTP Synced (Yes), Globally Registered (No), Node Role (Test Host), Access Policy (R&E Only, Access Notes), Virtual Machine (Yes), RAM (4 GB), and More Info (Details). Below these are 'Communities' (ESNet, UofSC) and 'On-demand testing tools' (Reverse ping, Reverse traceroute).

SERVICE	STATUS	VERSION	PORTS	SERVICE LOGS
esmond	Running	2.1.3-1.e17		View
lsregistration	Running	4.1.6-1.e17		View
owamp	Running	3.5.8-1.e17	861	View
pscheduler	Running	1.1.6-2.e17		View
psconfig	Running	4.1.6-1.e17		View
twamp	Running	3.5.8-1.e17	862	View

This concludes Lab 1.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swamy and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 2: perfSONAR Metrics and Tools

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	4
Lab roadmap.....	4
1 Introduction.....	4
2 Throughput measurement tools.....	5
2.1 iperf3.....	5
2.2 Nuttcp	7
3 Latency measurement tools	9
3.1 Ping	9
3.2 Owping.....	10
4 Trace test	12
4.1 Traceroute.....	13
4.2 Tracepath	14
4.3 Paris traceroute	15
References.....	18

Overview

This lab introduces the reader to network metrics using perfSONAR tools. It also explains how to use perfSONAR tools to measure the parameters that can affect the performance of networks.

Objectives

By the end of this lab, the user will:

1. Understand about network metrics.
2. Perform measurement test using perfSONAR tools.
3. Comprehend measurement results.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled as perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

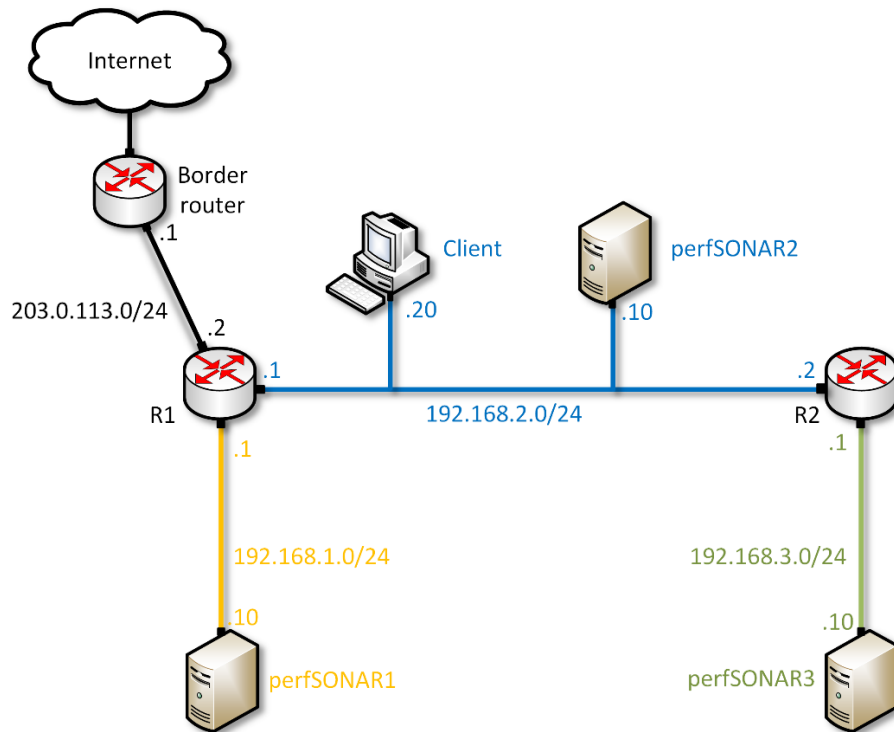


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

1. Section 1: Introduction.
2. Section 2: Throughput Measurement Tools.
3. Section 3: Latency Measurement Tools.
4. Section 4: Trace Tools.

1 Introduction

The only effective way to qualify and quantify the usage and behavior of a network is performing measurement tests. Knowing the network behavior is critical to diagnose network problems and performance issues. Metrics are quantitative and qualitative way to verify if a network achieves a desired behavior. Network managers are interested to measure the performance or availability of services. Therefore, the most typical metrics are connectivity, latency, packet loss rate, bandwidth and throughput. These metrics are introduced as follows:

- *Connectivity*: It determines whether two hosts can establish a connection between each other through the network.
- *Latency*: It is the time it takes for a packet to arrive form the source node to the destination host. Latency is also referred as network delay and can be measured in one-way or two-way latency. Two-way delay is also known as Round-Trip Time (RTT).
- *Packet loss rate*: It is the rate at which packets are being lost in their transit from the source to the destination host. Packets being lost means that the packet does not arrive to the intended destination.

- *Bandwidth*: Depending on the context, the term is used to describe either the physical link capacity in terms of signaling or the maximum actual data rate of a specific network link or a path can transfer.
- *Throughput*: It is a measure for amount of data being transferred across a link or network at a certain time.

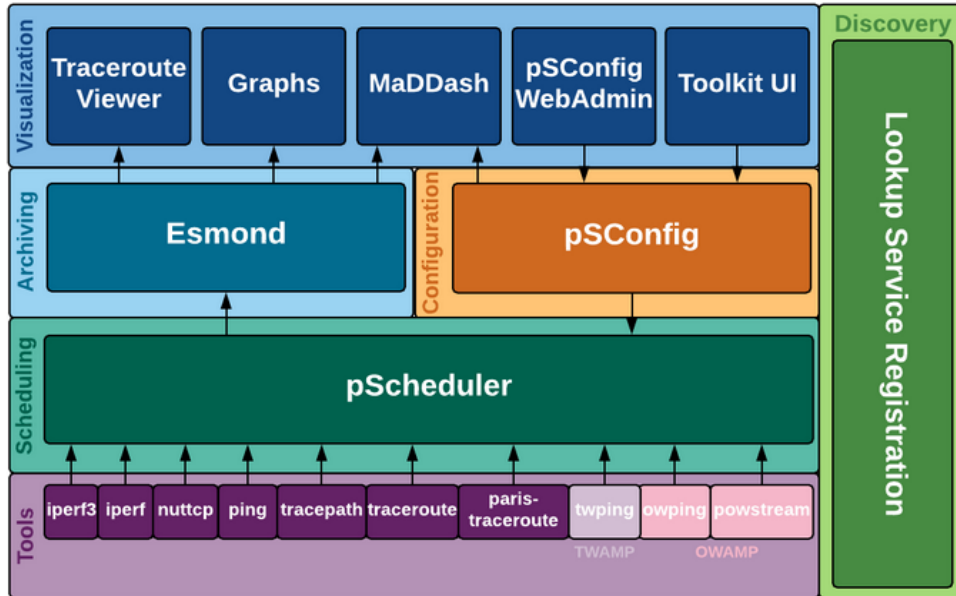


Figure 2. perfSONAR layers⁵.

perfSONAR tools provide the test to measure network metrics. These tools can be combined to provide a picture of the capabilities of a network. This lab is aimed to provide a brief description about the tools used by perfSONAR to run throughput, latency and trace measurements. These tests are delivered by the *Tools* layer as shown in the figure 2. In the following sections the user will use the tools available in the *Tool* layer to run throughput, latency and trace tests.

2 Throughput measurement tools

In this section the user will run measurement tests using *iperf3* and *nuttcp*. By default, perfSONAR uses *iperf3*. These tools are used by perfSONAR to measure the throughput. First, the user will run throughput tests using *iperf3* commands. Secondly, the user will measure the throughput using *nuttcp* commands. Finally, there is a brief analysis about the differences between the *tools*.

2.1 iperf3

iperf3 is a real-time network throughput measurement tool. It is an open source and cross-platform client-server application that can be used to measure the throughput between the two end devices. Typical *iperf3* output contains a time-stamped report of the amount of data transferred and the measured throughput.

The user interacts with *iperf3* using the `iperf3` command. The basic *iperf3* syntax used on both the client and the server is as follows:

```
iperf3 [-s|-c] [options]
```

Step 1. Open perfSONAR1 and enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.10.1.el7.x86_64 on an x86_64

perfsonar1 login: admin
Password:
Last login: Mon Apr 15 10:12:00 on tty1
Welcome to the perfSONAR Toolkit v4.1.6-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar1 ~]$_
```

Step 2. To launch *iperf3* in server mode, run the command `iperf3 -s` in perfSONAR1 command line. The parameter `-s` in the command above indicates that the host is configured as a server. Now, the server is listening on port 5201 waiting for incoming connections.

```
[admin@perfsonar1 ~]$_ iperf3 -s
-----
Server listening on 5201
-----
```

Step 3. Open perfSONAR2 and enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar2 login: admin
Password:
Last login: Mon Apr 15 10:31:27 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar2 ~]$_
```

Step 4. Now to launch *iperf3* in client mode, run the command `iperf3 -c 192.168.1.10` in perfSONAR2 node. The parameter `-c` in command above indicates that the host is

configured as an *iperf3* client. The parameter *192.168.1.10* is the IP address of the server in this case, perfSONAR1 node.

```

[admin@perfsonar2 ~]$ iperf3 -c 192.168.1.10
Connecting to host 192.168.1.10, port 5201
[ 51] local 192.168.2.10 port 48698 connected to 192.168.1.10 port 5201
[ ID] Interval          Transfer          Bitrate          Retr    Cwnd
[ 51] 0.00-1.00 sec      716 MBytes      6.00 Gbits/sec   1329    635 KBytes
[ 51] 1.00-2.00 sec      696 MBytes      5.84 Gbits/sec    96     624 KBytes
[ 51] 2.00-3.00 sec      684 MBytes      5.73 Gbits/sec   124     752 KBytes
[ 51] 3.00-4.00 sec      689 MBytes      5.78 Gbits/sec   168     611 KBytes
[ 51] 4.00-5.00 sec      740 MBytes      6.21 Gbits/sec    89     619 KBytes
[ 51] 5.00-6.00 sec      741 MBytes      6.22 Gbits/sec   171     595 KBytes
[ 51] 6.00-7.00 sec      725 MBytes      6.08 Gbits/sec    16     829 KBytes
[ 51] 7.00-8.00 sec      698 MBytes      5.85 Gbits/sec   131     452 KBytes
[ 51] 8.00-9.00 sec      658 MBytes      5.52 Gbits/sec   127     568 KBytes
[ 51] 9.00-10.00 sec     694 MBytes      5.82 Gbits/sec    90     827 KBytes
-----
[ ID] Interval          Transfer          Bitrate          Retr
[ 51] 0.00-10.00 sec    6.87 GBytes      5.91 Gbits/sec   2341
[ 51] 0.00-10.04 sec    6.85 GBytes      5.87 Gbits/sec
iperf Done.
[admin@perfsonar2 ~]$ _

```

Once the test is completed, a summary report on both the client and the server is displayed containing the following data:

- *ID*: identification number of the connection.
- *Interval*: time interval to periodically report throughput. By default, the time interval is 1 second.
- *Transfer*: how much data was transferred in each time interval.
- *Bitrate*: the measured throughput in each time interval.
- *Retr*: the number of TCP segments retransmitted in each time interval. This field increases when TCP segments are lost in the network due to congestion or corruption.
- *Cwnd*: indicates the congestion windows size in each time interval. TCP uses this variable to limit the amount of data the TCP client can send before receiving the acknowledgement of the sent data.

The summarized data, which starts after the last dashed line, shows the total amount of transferred data 6.87 GBytes and the throughput 5.89 Gbps. Note that the results may vary.

Step 5. In order to stop the server, go back to perfSONAR1 CLI and press `Ctrl+c`. The user will see the throughput results in the server side too. The summarized data on the server is similar to the client side and must be interpreted in the same way.

2.2 Nuttcp

nuttcp is a network performance measurement tool intended for use by network and system managers. Its most basic usage is to determine the raw TCP/UDP network layer throughput by transferring memory buffers from a source system across an interconnecting network to a destination system, either transferring data for a specified

time interval, or alternatively transferring a specified number of bytes. In addition to reporting the achieved network throughput, *nuttcp* also provides additional useful information related to the data transfer such as user, system, and wall-clock time, transmitter and receiver CPU utilization, and loss percentage for UDP transfers.

The user interacts with *nuttcp* using the `nuttcp` command. The basic *nuttcp* syntax used on both the client and the server is as follows:

```
nuttcp [options] dest_IP
```

Step 1. To launch *nuttcp* in server mode, run the command `nuttcp -S` in perfSONAR1 CLI as shown in the figure below.

```
nuttcp -S
```

```
[admin@perfsonar1 ~]# nuttcp -S
[admin@perfsonar1 ~]# _
```

Step 2. To launch *nuttcp* in client mode, run the command shown below in perfSONAR2 CLI. The parameter `-i1` indicates the time interval for the results will be every 1 second. The parameter `192.168.1.10` is the IP address of the server perfSONAR1.

```
nuttcp -i1 192.168.1.10
```

```
[admin@perfsonar2 ~]# nuttcp -i1 192.168.1.10
669.3750 MB / 1.00 sec = 5615.1189 Mbps    16 retrans    4155 KB-cwnd
611.6875 MB / 1.00 sec = 5131.1759 Mbps    179 retrans    2080 KB-cwnd
641.2500 MB / 1.00 sec = 5378.8937 Mbps    189 retrans    770 KB-cwnd
659.0000 MB / 1.00 sec = 5528.4244 Mbps    54 retrans    784 KB-cwnd
649.5000 MB / 1.00 sec = 5448.2701 Mbps    168 retrans    766 KB-cwnd
672.0625 MB / 1.00 sec = 5637.8154 Mbps    151 retrans    757 KB-cwnd
676.3750 MB / 1.00 sec = 5673.8391 Mbps    36 retrans    776 KB-cwnd
653.8750 MB / 1.00 sec = 5485.0133 Mbps    152 retrans    731 KB-cwnd
613.0000 MB / 1.00 sec = 5142.1961 Mbps    54 retrans    691 KB-cwnd
610.0625 MB / 1.00 sec = 5117.6775 Mbps    54 retrans    677 KB-cwnd

6485.4513 MB / 10.04 sec = 5416.6847 Mbps 22 %TX 22 %RX 1053 retrans 684 KB-cwnd 0.31 msRTT
[admin@perfsonar2 ~]#
```

Once the test is completed, a summary report just on the client. Each line contains the following data:

- *Transferred Data*: how much data was transferred in each time interval.
- *Time Interval*: how long it takes between each transferred data.
- *Bitrate*: the measured throughput in in each time interval.
- *Retransmissions*: the number of TCP segments retransmitted in each time interval. This field increases when TCP segments are lost in the network due to congestion or corruption.
- *Congestion Window*: indicates the congestion windows size in each time interval. TCP uses this variable to limit the amount of data the TCP client can send before receiving the acknowledgement of the sent data.

The summarized data indicate that 6485.4513 MBytes were transferred in 10.04 seconds. This is equivalent to 5416.6847 Mbps. The results also show the CPU usage which in this case is 22% for either the transmitter (TX) and the receiver (RX). The number of retransmissions is 1053, the mean size of congestion windows is 684 KBytes and the Round-Trip Time (RTT) is 0.31ms.

Step 3. To stop the server, go back to perfSONAR1 CLI and type the command `kill nuttcp`.

```
admin@perfsonar1 ~]# kill nuttcp
admin@perfsonar1 ~]# _
```

The main differences between *iperf3* and *nuttcp* are that *nuttcp* also measures the CPU usage and Round-Trip Time (RTT). However, in *nuttcp* the user only sees the test report in the client side.

3 Latency measurement tools

perfSONAR uses *ping* and *owping* to measure the latency. By default, perfSONAR uses *ping* to measure the latency. In the following sections, the user will measure the latency using *ping* command. Then, the user will use *owping* command. Finally, there is a brief analysis about the differences between the *tools*.

3.1 Ping

The *ping* command sends Internet Control Message Protocol (ICMP) echo request messages to the destination computer and waiting for a response. The number of messages returned to the requester is a key factor to measure the round-trip time and the packet loss. perfSONAR uses this tool to measure both. In addition, this command is also useful to test the connectivity. The basic syntax of ping is as follows:

```
ping [options] dest_IP
```

Step 1. In order to run a *ping* test, in perfSONAR1 CLI, type the command shown below. The parameter `-c 10` indicates how many packets are going to be sent to the destination host. The destination IP address is *192.168.2.10*.

```
ping -c 10 192.168.2.10
```

```

[admin@perfsonar1 ~]$ ping -c 10 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=63 time=0.442 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=63 time=0.408 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=63 time=0.376 ms
64 bytes from 192.168.2.10: icmp_seq=4 ttl=63 time=0.384 ms
64 bytes from 192.168.2.10: icmp_seq=5 ttl=63 time=0.399 ms
64 bytes from 192.168.2.10: icmp_seq=6 ttl=63 time=0.383 ms
64 bytes from 192.168.2.10: icmp_seq=7 ttl=63 time=0.373 ms
64 bytes from 192.168.2.10: icmp_seq=8 ttl=63 time=0.399 ms
64 bytes from 192.168.2.10: icmp_seq=9 ttl=63 time=0.367 ms
64 bytes from 192.168.2.10: icmp_seq=10 ttl=63 time=0.404 ms

--- 192.168.2.10 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.367/0.393/0.442/0.028 ms
[admin@perfsonar1 ~]$ _

```

The result above indicates that all ten packets were received successfully by perfSONAR2 node (192.168.2.10) (0% packet loss) and that the minimum, mean, maximum, and standard deviation of the Round-Trip Time (RTT) were 0.367, 0.393, 0.442 and 0.028 milliseconds respectively.

Step 2. In order to run a *ping* test, in perfSONAR1 CLI, type the command shown below. The parameter `-c 10` indicates how many packets are going to be sent to the destination host. The destination IP address is *192.168.3.10*.

```
ping -c 10 192.168.3.10
```

```

[admin@perfsonar1 ~]$ ping -c 10 192.168.3.10
PING 192.168.3.10 (192.168.3.10) 56(84) bytes of data.
64 bytes from 192.168.3.10: icmp_seq=1 ttl=62 time=0.632 ms
64 bytes from 192.168.3.10: icmp_seq=2 ttl=62 time=0.513 ms
64 bytes from 192.168.3.10: icmp_seq=3 ttl=62 time=0.458 ms
64 bytes from 192.168.3.10: icmp_seq=4 ttl=62 time=0.503 ms
64 bytes from 192.168.3.10: icmp_seq=5 ttl=62 time=0.479 ms
64 bytes from 192.168.3.10: icmp_seq=6 ttl=62 time=0.535 ms
64 bytes from 192.168.3.10: icmp_seq=7 ttl=62 time=0.544 ms
64 bytes from 192.168.3.10: icmp_seq=8 ttl=62 time=0.551 ms
64 bytes from 192.168.3.10: icmp_seq=9 ttl=62 time=0.623 ms
64 bytes from 192.168.3.10: icmp_seq=10 ttl=62 time=0.554 ms

--- 192.168.3.10 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 0.458/0.539/0.632/0.055 ms
[admin@perfsonar1 ~]$ _

```

The result above indicates that all ten packets were received successfully by perfSONAR3 node (192.168.3.10) (0% packet loss) and that the minimum, mean, maximum, and standard deviation of the round-trip time (RTT) were 0.458, 0.539, 0.632 and 0.055 milliseconds respectively.

3.2 Owping

The *owping* is a command line client application and a policy daemon used to determine one-way latencies between hosts. With roundtrip-based measurements, it is hard to isolate the direction in which congestion is experienced. One-way measurements solve this problem and make the direction of congestion immediately apparent. Since traffic can be asymmetric at many sites that are primarily producers or consumers of data, this

allows for more informative measurements. One-way measurements allow the user to better isolate the effects of specific parts of a network on the treatment of traffic¹.

The basic syntax of *owping* is as follows:

```
owping [options] dest_IP
```

Step 1. In perfSONAR1 command line type the command shown below. The destination IP address is *192.168.2.10*. The results are going to be displayed after approximately 12.9 seconds.

```
owping 192.168.2.10
```

```
[admin@perfsonar1 ~]# owping 192.168.2.10
Approximately 12.9 seconds until results available

--- owping statistics from [perfsonar1]:9278 to [192.168.2.10]:9680 ---
SID:      c0a8020ae06503fa4d47fa66e7966599
first:    2019-04-19T22:35:39.326
last:     2019-04-19T22:35:48.027
100 sent, 0 lost (0.000%), 0 duplicates
one-way delay min/median/max = 0.129/0.4/137 ms, (err=0.751 ms)
one-way jitter = 1.7 ms (P95-P50)
hops = 1 (consistently)
no reordering

--- owping statistics from [192.168.2.10]:9311 to [perfsonar1]:8997 ---
SID:      c0a8010ae06503fa52cbc4342ff6fb46
first:    2019-04-19T22:35:39.253
last:     2019-04-19T22:35:48.588
100 sent, 0 lost (0.000%), 0 duplicates
one-way delay min/median/max = 0.0925/0.3/167 ms, (err=0.751 ms)
one-way jitter = 0.6 ms (P95-P50)
hops = 1 (consistently)
no reordering

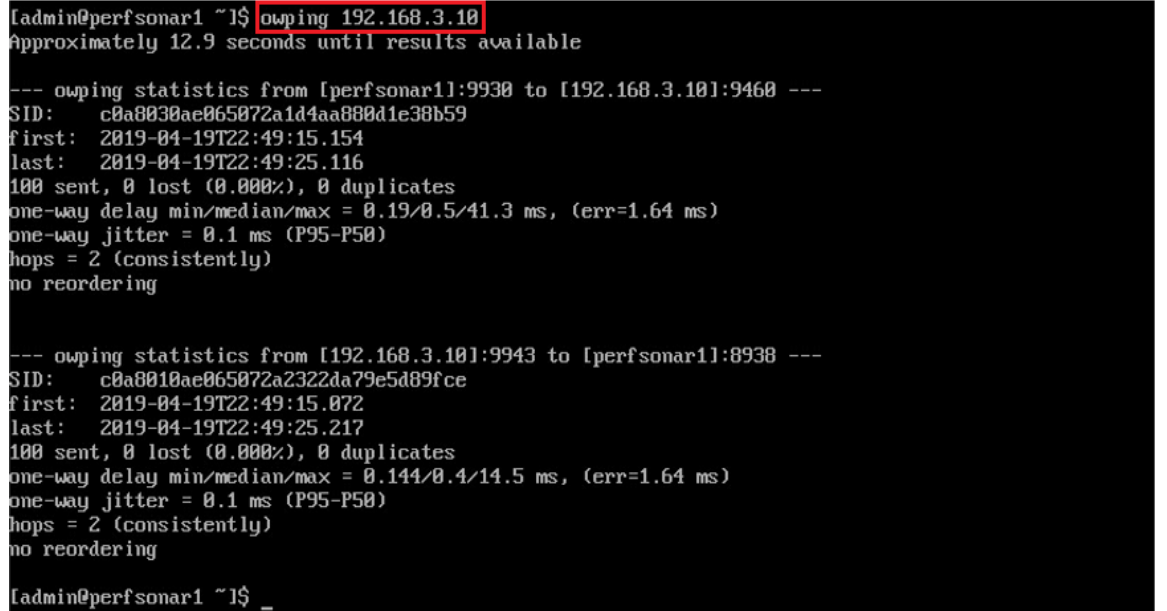
[admin@perfsonar1 ~]#
```

The figure above indicates results from perfSONAR1 (*192.168.1.10*) to perfSONAR2 (*192.168.2.10*). The source and destination ports number are 9278 and 9680 respectively. All packets were received successfully by perfSONAR2 node (*192.168.2.10*) (0% packet loss). The minimum, median and maximum one-way latency values were 0.129, 0.4 and 137 milliseconds respectively. The one-way jitter is 1.7 milliseconds, it takes 1 hop to reach the destination consistently, which means there no other way get that destination. Finally, there is not packet reordering.

The test result from perfSONAR2 (*192.168.2.10*) to perfSONAR1 (*192.168.1.10*) are similar to the previous one. In this case, the source and destination ports are 9311 and 8997 respectively. All packets were received successfully by perfSONAR1 node (*192.168.1.10*) (0% packet loss). The minimum, median and maximum one-way latency values were 0.0925, 0.3 and 167 milliseconds respectively. The one-way jitter is 0.1 milliseconds, it takes 1 hop to reach out the destination and there is not packet reordering.

Step 2. In perfSONAR1 command line type the command shown below. The destination IP address is *192.168.3.10*. The results are going to be displayed after approximately 12.9 seconds.

```
owping 192.168.3.10
```



```
[admin@perfsonar1 ~]$ owping 192.168.3.10
Approximately 12.9 seconds until results available

--- owping statistics from [perfsonar1]:9930 to [192.168.3.10]:9460 ---
SID:      c8a8030ae065072a1d4aa880d1e38b59
first:    2019-04-19T22:49:15.154
last:     2019-04-19T22:49:25.116
100 sent, 0 lost (0.000%), 0 duplicates
one-way delay min/median/max = 0.19/0.5/41.3 ms, (err=1.64 ms)
one-way jitter = 0.1 ms (P95-P50)
hops = 2 (consistently)
no reordering

--- owping statistics from [192.168.3.10]:9943 to [perfsonar1]:8938 ---
SID:      c8a8010ae065072a2322da79e5d89fce
first:    2019-04-19T22:49:15.072
last:     2019-04-19T22:49:25.217
100 sent, 0 lost (0.000%), 0 duplicates
one-way delay min/median/max = 0.144/0.4/14.5 ms, (err=1.64 ms)
one-way jitter = 0.1 ms (P95-P50)
hops = 2 (consistently)
no reordering

[admin@perfsonar1 ~]$ _
```

The results shown above are similar to the previous one. In this case, test results are from perfSONAR1 (*192.168.1.10*) to perfSONAR3 (*192.168.3.10*). The source and destination ports number are 9930 and 9460 respectively. All packets were received successfully by perfSONAR3 node (*192.168.3.10*) (0% packet loss). The minimum, median and maximum one-way latency values were 0.19, 0.5, 41.3 milliseconds. The one-way jitter is 0.1 milliseconds, it takes 2 hops to reach out the destination and there is not packet reordering.

By the other hand, test results from perfSONAR3 (*192.168.3.10*) to perfSONAR1 (*192.168.1.10*) are like the first one. In this case, the source and destination ports number are 9943 and 8938 respectively. All packets were received successfully by perfSONAR1 node (*192.168.1.10*) (0% packet loss). The minimum, median and maximum one-way latency values were 0.144., 0.4, 14.5 milliseconds respectively. The one-way jitter is 0.1 milliseconds, it takes 2 hops to reach out the destination and there is not packet reordering.

4 Trace test

Trace tests are networking tools which allow to discover the path a data packet takes to go from a source node to a destination node. The trace tools which perfSONAR uses are *traceroute*, *tracpath* and *paris-tracpath*. In this section, the user will run trace tests using these tools. By default, perfSONAR uses *traceroute*.

4.1 Traceroute

Traceroute displays the path that a packet took as it traveled through the network. It also displays times which are the response times that occurred at each stop along the route. If there is a connection problem or latency connecting to a site, it will be perceived analyzing these times. The user will be able to identify which of the hops along the route may cause a problem³.

The basic syntax of *traceroute* is as follows:

```
traceroute[options] dest_IP
```

Step 1. In perfSONAR1 command line type the command shown below. The IP address of the destination is *192.168.2.10*.

```
traceroute 192.168.2.10
```

```
[admin@perfsonar1 ~]# traceroute 192.168.2.10
traceroute to 192.168.2.10 (192.168.2.10), 30 hops max, 60 byte packets
 1 gateway (192.168.1.1)  0.373 ms  0.252 ms  0.182 ms
 2 192.168.2.10 (192.168.2.10)  0.439 ms  0.407 ms  0.302 ms
[admin@perfsonar1 ~]# _
```

In the figure above, there are several rows divided into columns on the report. Each row represents a hop along the route. In each hope, the packet gets its next set of directions. Each row is divided into five columns. A sample row is shown below:

HOP NUMBER	IP ADDRESS	RTT 1	RTT 2	RTT 3
1	192.168.1.1	0.373 ms	0.252 ms	0.182 ms
2	192.168.2.10	0.439 ms	0.407 ms	0.302 ms

- HOP NUMBER: It represents the number of the hop along the route. In this case, it takes two hops to reach out the destination.
- IP ADDRESS: The second column has the IP address of the destination; the previous hope has the IP address of the router. If it is available, the domain name will also be listed.
- RTT Columns: The next three columns display the round-trip time (RTT) for the packet to reach that point and return to the source host. This measure is listed in milliseconds. There are three columns because the traceroute sends three separate signal packets. This is to display consistency, or a lack thereof, in the route.

Step 2. In perfSONAR1 command line type the command shown below. The IP address of the destination is *192.168.3.10*.

```
traceroute 192.168.3.10
```

```
[admin@perfsonar1 ~]# traceroute 192.168.3.10
traceroute to 192.168.3.10 (192.168.3.10), 30 hops max, 60 byte packets
 1 gateway (192.168.1.1)  0.383 ms  0.286 ms  0.228 ms
 2 192.168.2.2 (192.168.2.2)  0.577 ms  0.551 ms  0.525 ms
 3 192.168.3.10 (192.168.3.10)  0.677 ms  0.649 ms  0.631 ms
[admin@perfsonar1 ~]#
```

In the figure above, there are several rows divided into columns on the report. Each row represents a hop along the route. In each hope, the packet gets its next set of directions. Each row is divided into five columns. A sample row is shown below:

HOP NUMBER	IP ADDRESS	RTT 1	RTT 2	RTT 3
1	192.168.1.1	0.383 ms	0.286 ms	0.228 ms
2	192.168.2.2	0.557 ms	0.551 ms	0.525 ms
3	192.168.3.10	0.677 ms	0.649 ms	0.631 ms

- **HOP NUMBER:** It represents the number of the hop along the route. In this case, it takes three hops to reach out the destination.
- **IP ADDRESS:** The second column has the IP address of the destination; the previous hope has the IP address of the router. If it is available, the domain name will also be listed.
- **RTT Columns:** The next three columns display the round-trip time (RTT) for the packet to reach that point and return to the source host. This measure is listed in milliseconds. There are three columns because the traceroute sends three separate signal packets. This is to display consistency, or a lack thereof, in the route.

4.2 Tracepath

This tool traces a path from the source to destination discovering the Maximum Transmission Unit (MTU) along this path. It uses UDP port or some random port. The difference with *traceroute* is that this tool includes less options and it is not required to be a superuser to run the tests.

The basic syntax of *tracepath* is as follows:

```
tracepath[options] dest_IP
```

Step 1. In perfSONAR1 command line type the command. The IP address of the destination is *192.168.2.10*.

```
tracepath 192.168.2.10
```

```
[admin@perfsonar1 ~]# tracepath 192.168.2.10
1?: [LOCALHOST] pmtu 1500
 1: gateway 0.336ms
 1: gateway 0.152ms
 2: 192.168.2.10 0.353ms !H
Resume: pmtu 1500
[admin@perfsonar1 ~]#
```


The first column shows the hop number, the second column shows the IP address or Domain name. The third column shows the Round-Trip Time (RTT) for the packet to reach that point and return to the source host. At the end a resume is displayed, in this case the user will see the Path MTU.

Step 2. In perfSONAR1 command line type the command. The IP address of the destination is *192.168.3.10*.

```
tracert 192.168.3.10
```

```
[admin@perfsonar1 ~]# tracert 192.168.3.10
 1?: [LOCALHOST]                pmtu 1500
 1: gateway                      0.337ms
 1: gateway                      0.158ms
 2: 192.168.2.2                  0.488ms
 3: 192.168.3.10                 0.460ms !H
Resume: pmtu 1500
[admin@perfsonar1 ~]#
```

Similarly, the first column shows the hop number, the second column shows the IP address or Domain name. The third column shows the round-trip time (RTT) for the packet to reach that point and return to the source host. At the end a resume is displayed, in this case the user will see the Path MTU.

4.3 Paris traceroute

Paris traceroute is a new version of the *traceroute* network diagnosis tool. It addresses problems caused by load balancers with the initial traceroute implementation. By controlling the flow identifier of the probes, it is able to follow accurate paths in networks with load balancers. It is also able to find all the load balanced paths to the destination. Finally, it complements its output with information extracted from the received packets, allowing a more precise analysis of the discovered paths. Paris traceroute, by controlling packet header contents, obtains a more precise picture of the actual routes that packets follow⁴.

To exemplify this idea, a topology is presented in the leftmost portion of figure 3, where A is a router that balances load across two paths, via routers B or C. The middle figure illustrates what the result might show with classic traceroute. The figure on the right is the result using Paris traceroute.

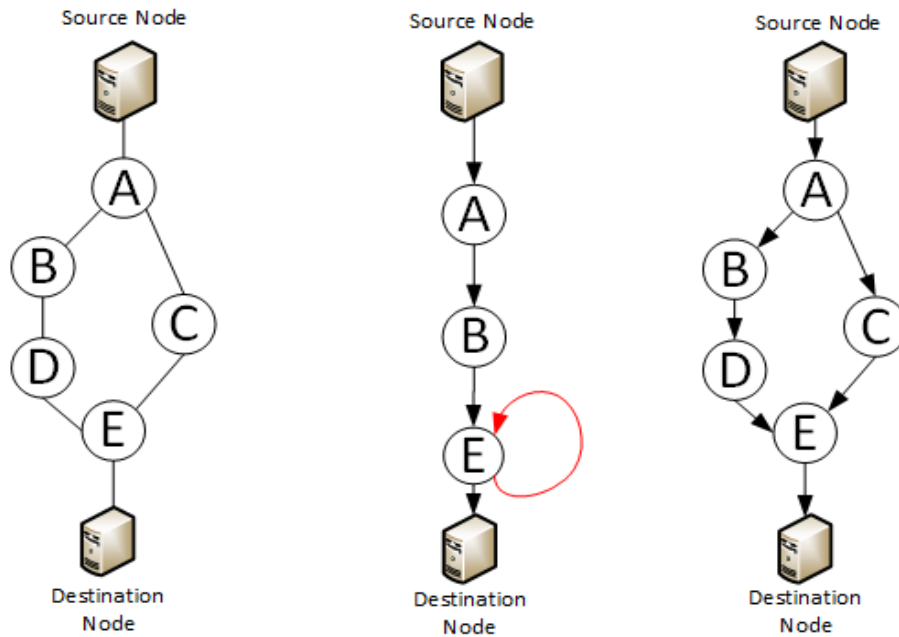


Figure 3. Paris traceroute behavior.

The basic syntax of traceroute is as follows:

```
paris-traceroute [options] dest_IP
```

Step 1. In perfSONAR1 command line type the command shown below. The IP address of the destination is `192.168.2.10`. The user may be required to authenticate, in that case type `admin` as the password, notice that the password will not be displayed while typing it.

```
sudo paris-traceroute 192.168.2.10
```

```
[admin@perfsonar1 ~]# sudo paris-traceroute 192.168.2.10
[sudo] password for admin:
traceroute to 192.168.2.10 (192.168.2.10), 30 hops max, 30 bytes packets
 1 gateway (192.168.1.1) 1.076ms  1.074ms  1.076ms
 2 192.168.2.10 (192.168.2.10) 0.319ms  0.306ms  0.301ms
[admin@perfsonar1 ~]# _
```

Notice that it is not possible to prove the concept of Paris traceroute in the current lab topology.

The results of Paris-traceroute test are interpreted in the same way of traceroute test. In the figure above, there are several rows divided into columns on the report. These results are reordered in the table below. Each row represents a hop along the route. In each hope, the packet gets its next set of directions. Each row is divided into five columns. A sample row is shown below:

HOP NUMBER	IP ADDRESS	RTT 1	RTT 2	RTT 3
1	192.168.1.1	1.076 ms	1.074 ms	1.076 ms
2	192.168.2.10	0.319 ms	0.306 ms	0.301 ms

- **HOP NUMBER:** It represents the number of the hop along the route. In this case, it takes two hops to reach out the destination.
- **IP ADDRESS:** The second column has the IP address of the destination; the previous hope has the IP address of the router. If it is available, the domain name will also be listed.
- **RTT Columns:** The next three columns display the round-trip time (RTT) for the packet to reach that point and return to the source host. This measure is listed in milliseconds. There are three columns because the traceroute sends three separate signal packets. This is to display consistency, or a lack thereof, in the route.

Step 2. In perfSONAR1 command line type the command shown below. The IP address of the destination is `192.168.3.10` The user may be required to authenticate, in that case type `admin` as the password, notice that the password will not be displayed while typing it.

```
sudo paris-traceroute 192.168.3.10
```

```

[admin@perfsonar1 ~]$ sudo paris-traceroute 192.168.3.10
traceroute to 192.168.3.10 (192.168.3.10), 30 hops max, 30 bytes packets
 1 gateway (192.168.1.1) 1.028ms 1.023ms 1.022ms
 2 192.168.2.2 (192.168.2.2) 0.357ms 10.135ms 10.136ms
 3 192.168.3.10 (192.168.3.10) 0.489ms 0.470ms 0.465ms
[admin@perfsonar1 ~]$
    
```

The results should be interpreted as the traceroute tool. In the figure above, there are several rows divided into columns on the report. Each row represents a hop along the route. In each hope, the packet gets its next set of directions. Each row is divided into five columns. A sample row is shown below:

HOP NUMBER	IP ADDRESS	RTT 1	RTT 2	RTT 3
1	192.168.1.1	1.028 ms	1.023 ms	1.022 ms
2	192.168.2.2	0.357 ms	10.135 ms	10.136 ms
3	192.168.3.10	0.489 ms	0.470 ms	0.465 ms

- **HOP NUMBER:** It represents the number of the hop along the route. In this case, it takes three hops to reach out the destination.
- **IP ADDRESS:** The second column has the IP address of the destination; the previous hope has the IP address of the router. If it is available, the domain name will also be listed.
- **RTT Columns:** The next three columns display the round-trip time (RTT) for the packet to reach that point and return to the source host. This measure is listed in milliseconds. There are three columns because the traceroute sends three separate signal packets. This is to display consistency, or a lack thereof, in the route.

This concludes Lab 2.

References

1. Internet2, "One-Way Ping Documentation," [Online]. Available: <https://software.internet2.edu/owamp/owampd.pfs.man.html>.
2. Inmotion Hosting, "How to Read a Traceroute," [Online]. Available: <https://www.inmotionhosting.com/support/website/how-to/read-traceroute>
3. Paris Traceroute, "Paris Traceroute," [Online]. Available: <https://paris-traceroute.net>.
4. NSRC, "What Is Perfsonar?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
5. perfSONAR, "Test and Tool Reference," [Online]. Available: http://docs.perfsonar.net/pscheduler_ref_tests_tools.html
6. ManKier, "iPerf Man Page," [Online]. Available: <https://www.mankier.com/1/iperf>.
7. Ping-Linux Man Page, [Online]. Available: <https://www.mankier.com/1/iperf>.
8. Traceroute-Linux Man Page, [Online]. Available: <https://linux.die.net/man/8/traceroute>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	4
Lab roadmap.....	4
1 Introduction.....	4
2 Configuring regular tests	5
2.1 Accessing the web user interface	5
2.2 Configuring throughput test	6
2.3 Configuring latency and packet loss tests.....	13
3 Configuring R1 and R2 to emulate a Wide Area Network (WAN).....	19
3.1 Adding delay to interface connecting to network 192.168.2.0/24.....	19
3.2 Adding packet loss to interface connecting to network 192.168.2.0/24.....	22
References.....	25

Overview

This lab introduces the reader to perfSONAR Toolkit. At the end of this lab, the user will configure regular tests using perfSONAR Toolkit Graphical User Interface (GUI) in a Wide Area Network (WAN).

Objectives

By the end of this lab, the user will:

1. Configure regular test using perfSONAR GUI.
2. Store measurement data.
3. Set the parameters of the tests.
4. Conduct regular tests and measure the performance on a WAN.
5. Visualize the measurement results.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

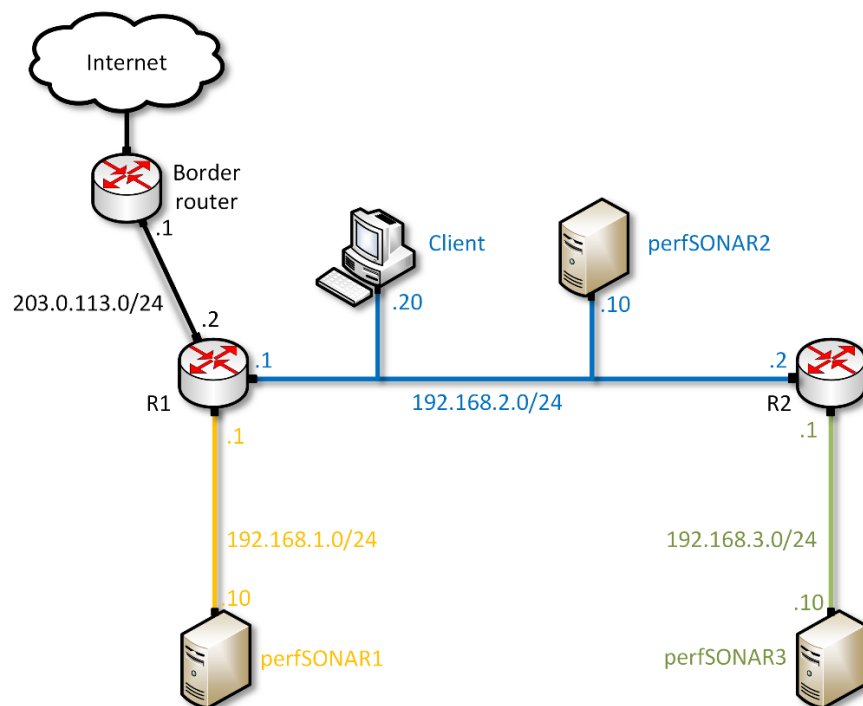


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes and the Client host.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Configuring regular tests.
3. Section 3: Configuring R1 and R2 to emulate a Wide Area Network (WAN).

1 Introduction

perfSONAR toolkit brings a web user interface to configure, manage and display test results as throughput, latency and packet loss. A core function of the perfSONAR Toolkit is to run regularly scheduled network measurements. The user can define the tests through the toolkit's graphical user interface (GUI).

perfSONAR Toolkit GUI is component the visualization layer as shown in the figure 2. perfSONAR includes many other utilities responsible for visualizing the measurement results. The user can configure regular tests using perfSONAR Toolkit GUI. In general, the user will not invoke any tools directly but, instead use the graphical user interface to execute them. perfSONAR Toolkit GUI interacts with the archiving layer which at the same time interacts with the scheduling layer that is responsible on deliver the requested test.

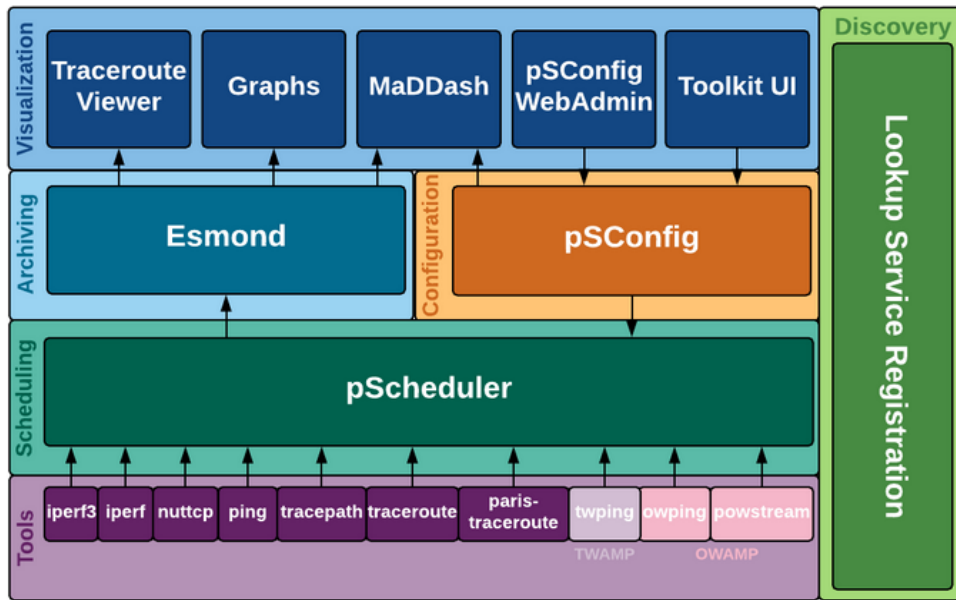


Figure 2. perfSONAR layers².

2 Configuring regular tests

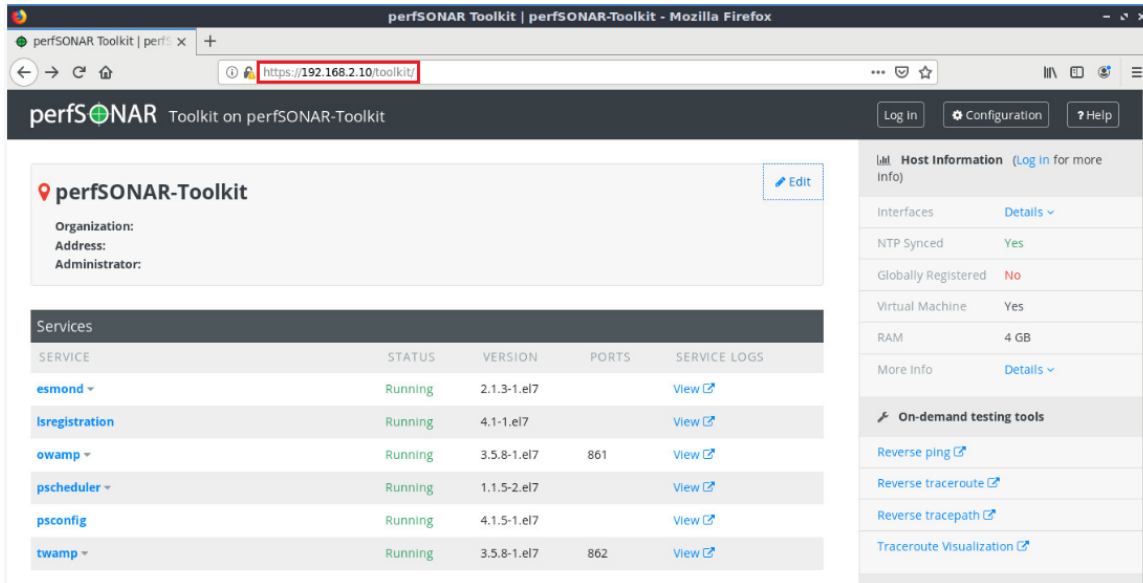
perfSONAR Toolkit provides the tools to run regularly scheduled network measurements. The user can define the tests which run through the toolkit's web interface. In this section the user will access the web user interface to run the measurement and storage of tests such as throughput, one-way ping and loss.

2.1 Accessing the web user interface

Step 1. On the Client host, open web browser located on the desktop.

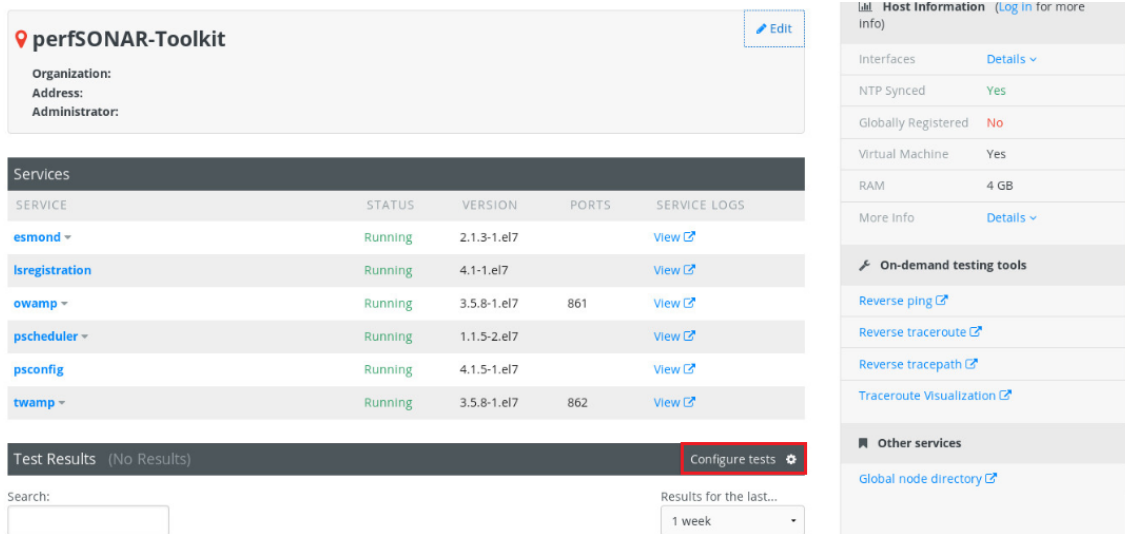


Step 2. On the address bar, type `192.168.2.10`. That is the IP address of perfSONAR2 toolkit node. The user will see the perfSONAR toolkit web interface.



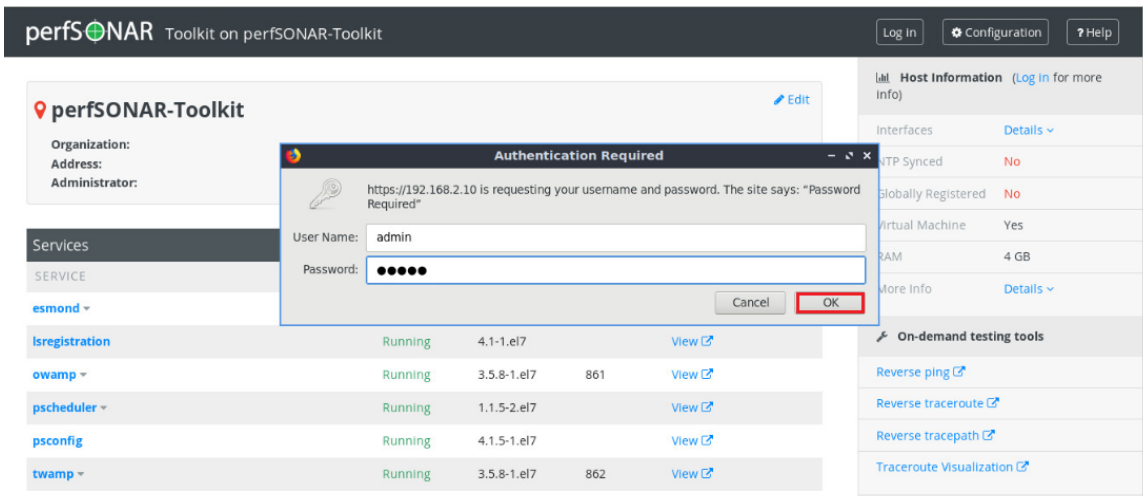
2.2 Configuring throughput test

Step 1. In the section Test Results, click on *Configure Test*.

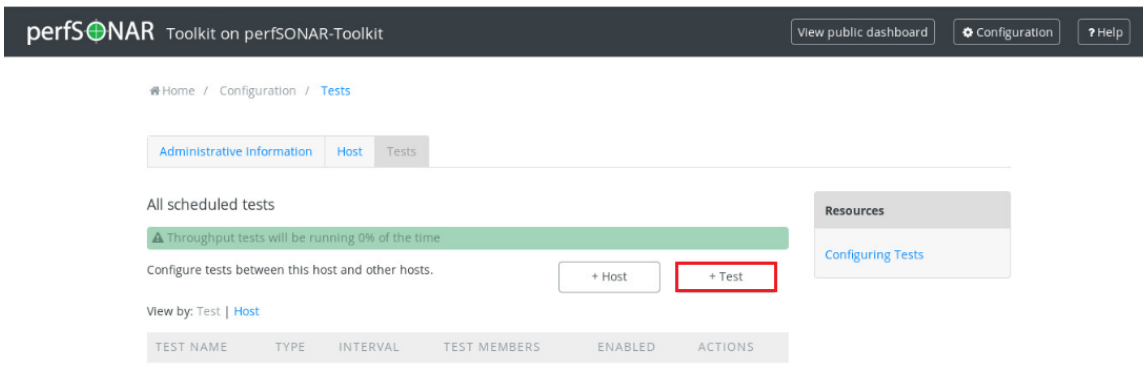


Step 2. In this step, the user could be required to authenticate. Type `admin` as the username, and `admin` as the password.

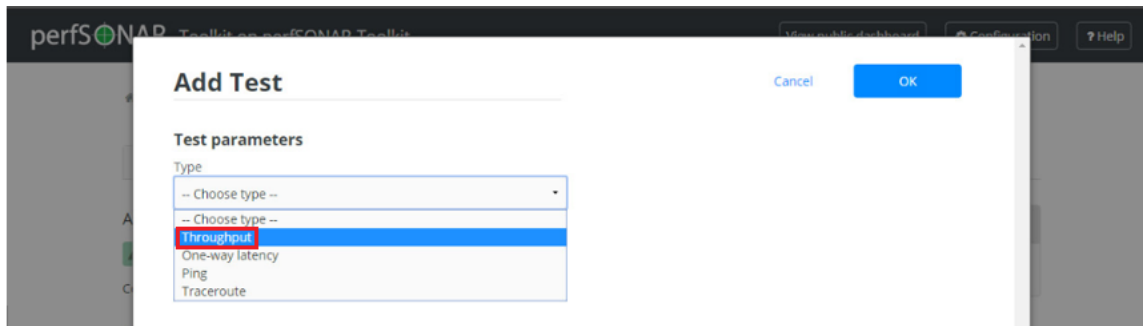
Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface



Step 3. Click on + *Test* to access to the test configuration form.



Step 4. A drop-down list shows to choose the test type. Select *Throughput* to proceed with the configuration.



Step 5. A new window will appear prompting the user for the parameters of the test. Type the *Test name/description* as *Throughput Test*.

Add Test Cancel OK

Test parameters

Type
Throughput

Test name/description
Throughput Test

Test Status
 Enabled

Interface
Default

Protocol
TCP

Time between tests
6

Units
Hours

Test duration
20

Units
Seconds

[+ Advanced Parameters](#)

Step 6. Select the interface *ens32*, notice that the perfSONAR2 node IP address will be displayed at the interface.

Add Test Cancel OK

Test parameters

Type
Throughput

Test name/description
Throughput Test

Test Status
 Enabled

Interface
ens32 - 192.168.2.10

Protocol
TCP

Time between tests
6

Units
Hours

Test duration
20

Units
Seconds

[+ Advanced Parameters](#)

Step 7. Set *Time between tests* to 1, and *Units* to *minutes*. and the duration of each test will be 20 seconds.

Add Test

Cancel

OK

Test parameters

Type

Throughput

Test name/description

Throughput Test

Test Status

Enabled

Interface

ens32 - 192.168.2.10

Protocol

TCP

Time between tests

Units

1

Minutes

Test duration

20

Units

Seconds

[+ Advanced Parameters](#)

In a production network, the time interval between tests is around 1 hour however, the user will set the interval between tests to 1 minute in order to have the results propagated.

Add Test

Cancel

OK

Test parameters

Type

Throughput

Test name/description

Throuput Test

Test Status

Enabled

Interface

ens32 - 192.168.2.10

Protocol

TCP

Time between tests

Units

1

Minutes

Test duration

20

Units

Seconds

[+ Advanced Parameters](#)

Step 8. Scroll down until *Test members* section. In *Hostname/IP* type the IP address of perfSONAR1 node *192.168.1.10*. In the *Host description* type *perfSONAR1*.

Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface

Interface: ens32 - 192.168.2.10 | Protocol: TCP

Time between tests: 1 | Units: Minutes | Test duration: 20 | Units: Seconds

+ Advanced Parameters

Test members

HOST	DESCRIPTION	IPV
------	-------------	-----

+ Add Test Member(s)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP	Host description	IPv4 <input checked="" type="checkbox"/>	IPv6 <input type="checkbox"/>
192.168.1.10	perfSONAR1		

Add host

Cancel | OK

Step 9. Click on *Add host* to save the changes

Interface: ens32 - 192.168.2.10 | Protocol: TCP

Time between tests: 1 | Units: Minutes | Test duration: 20 | Units: Seconds

+ Advanced Parameters

Test members

HOST	DESCRIPTION	IPV
------	-------------	-----

+ Add Test Member(s)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP	Host description	IPv4 <input checked="" type="checkbox"/>	IPv6 <input type="checkbox"/>
192.168.1.10	perfSONAR1		

Add host

Cancel | OK

Step 10. Similarly, in *Hostname/IP* type the IP address of perfSONAR3 node *192.168.3.10*. In the *Host description* type *perfSONAR3*.

Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface

Interface: ens32 - 192.168.2.10 | Protocol: TCP

Time between tests: 1 | Units: Minutes | Test duration: 20 | Units: Seconds

+ Advanced Parameters

Test members

HOST	DESCRIPTION	IPV
192.168.1.10	perfSONAR1	IPv4 <input checked="" type="checkbox"/> IPv6 <input type="checkbox"/>

+ Add Test Member(s)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: 192.168.3.10 | Host description: perfSONAR3 | IPv4 IPv6

Step 11. Click on *Add host* to save the changes.

Interface: ens32 - 192.168.2.10 | Protocol: TCP

Time between tests: 1 | Units: Minutes | Test duration: 20 | Units: Seconds

+ Advanced Parameters

Test members

HOST	DESCRIPTION	IPV
192.168.1.10	perfSONAR1	IPv4 <input checked="" type="checkbox"/> IPv6 <input type="checkbox"/>

+ Add Test Member(s)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: 192.168.3.10 | Host description: perfSONAR3 | IPv4 IPv6

Step 12. In order to save the changes, click on *OK*.

Test members

HOST	DESCRIPTION	IPV		
192.168.1.10	<input type="text" value="perfSONAR1"/>	IPv4 <input checked="" type="checkbox"/>	IPv6 <input type="checkbox"/>	
192.168.3.10	<input type="text" value="perfSONAR3"/>	IPv4 <input checked="" type="checkbox"/>	IPv6 <input type="checkbox"/>	

[+ Add Test Member\(s\)](#)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: Host description: IPv4 IPv6

Step 13. To save the test click on *Save*.

perfSONAR Toolkit on perfSONAR-Toolkit

Home / Configuration / Tests

Administrative Information
Host
Tests

All scheduled tests

Throughput tests will be running 0% of the time

Configure tests between this host and other hosts.

View by: Test | [Host](#)

TEST NAME	TYPE	INTERVAL	TEST MEMBERS	ENABLED	ACTIONS
Throughput Test	Throughput - TCP	1 minute	2 hosts	<input checked="" type="checkbox"/>	

Resources

[Configuring Tests](#)

You've made changes that haven't been saved.

Step 14. Click on *View public dashboard* to get back to main page and see the throughput results.

perfSONAR Toolkit on perfSONAR-Toolkit

Home / Configuration / Tests

Administrative Information
Host
Tests

All scheduled tests

Throughput tests will be running 233% of the time

Configure tests between this host and other hosts.

View by: Test | [Host](#)

TEST NAME	TYPE	INTERVAL	TEST MEMBERS	ENABLED	ACTIONS
Throughput Test	Throughput - TCP	1 minute	2 hosts	<input checked="" type="checkbox"/>	
perfSONAR Toolkit Default Traceroute Test	Traceroute	10 minutes	2 hosts	<input type="checkbox"/>	

Resources

[Configuring Tests](#)

Step 15. After 3 minutes the data will be propagated thus, refresh the browser and scroll down until the *Test Result* section.

The screenshot shows the perfSONAR GUI. At the top, there is a list of services: **owamp**, **pscheduler**, **psconfig**, and **twamp**. All are in a 'Running' state. Below this is a 'Test Results' section with 2 results. A search bar and a dropdown for 'Results for the last...' (set to 1 week) are present. The test results table is as follows:

SOURCE	DESTINATION	THROUGHPUT	LATENCY (MS)	LOSS
192.168.2.10 Details	192.168.3.10	→ 4.72 Gbps ← n/a	→ n/a ← n/a	→ n/a ← n/a
192.168.2.10 Details Traceroute	192.168.1.10	→ 4.79 Gbps ← n/a	→ n/a ← n/a	→ n/a ← n/a

At the bottom of the table, it says 'Showing 1 to 2 of 2 entries' with 'Previous' and 'Next' buttons.

This page is also available as [JSON](#)

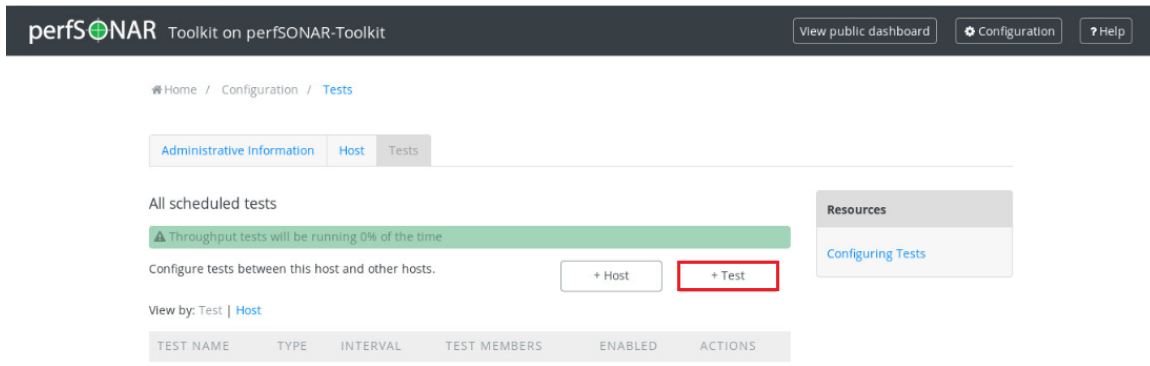
The result above indicates the throughput is 4.72 Gbps, when the source is *192.168.2.10* and the destination is *192.168.3.10*. Then, when the source is *192.168.2.10* and the destination is *192.168.3.10*, the throughput is 4.79 Gbps. Notice that the latency and loss results are not available. Note that the results may vary.

2.3 Configuring latency and packet loss tests

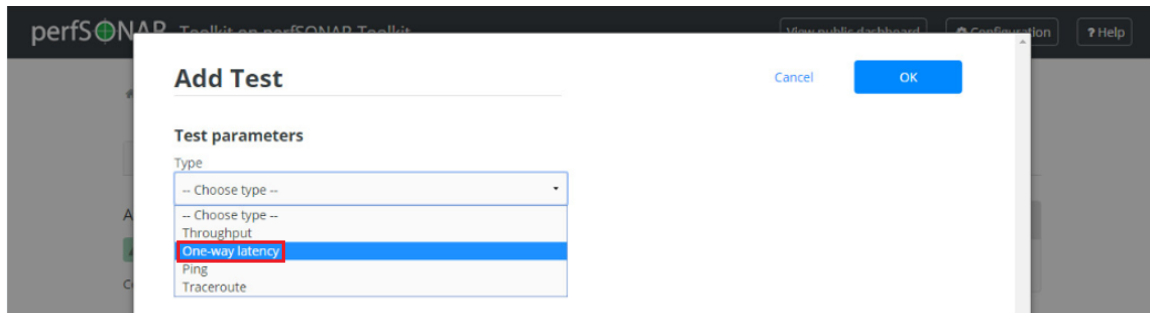
Step 1. In the section Test Results, click on *Configure Test*. In this step, the user could be required to login. If that the case, type `admin` as the username, and `admin` as the password.

The screenshot shows the perfSONAR-Toolkit GUI. On the left, there is a 'Services' table with columns: SERVICE, STATUS, VERSION, PORTS, and SERVICE LOGS. The services listed are **esmond**, **lsregistration**, **owamp**, **pscheduler**, **psconfig**, and **twamp**, all in a 'Running' state. Below this is a 'Test Results' section with '(No Results)' and a 'Configure tests' button. On the right, there is a 'Host Information' sidebar with sections: 'Host Information', 'On-demand testing tools' (containing Reverse ping, Reverse traceroute, Reverse tracepath, and Traceroute Visualization), and 'Other services' (containing Global node directory).

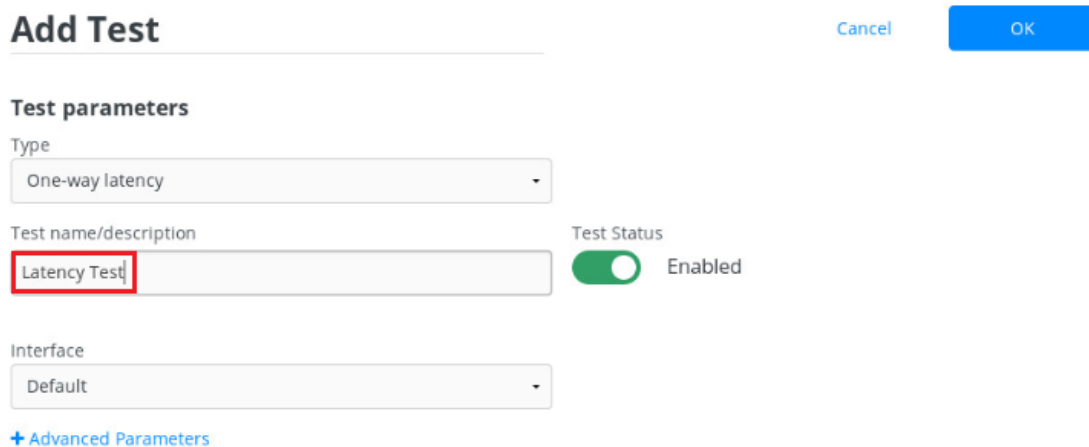
Step 2. Click on *+Test* to access to the test configuration form.



Step 3. A drop-down list shows to choose the test type. Select *One-way latency* to proceed with the configuration.



Step 5. A new window will appear prompting the user for the parameters of the test. Type the *Test name/description* as *Latency Test*.



Step 6. Select the interface *ens32*, notice that the perfSONAR2 node IP address will be displayed at the interface.

Add Test

Cancel

Test parameters

Type
One-way latency

Test name/description
Latency Test

Test Status
 Enabled

Interface
ens32 - 192.168.2.10

[+ Advanced Parameters](#)

Step 7. Scroll down until *Test members* section. In *Hostname/IP* type the IP address of perfSONAR1 node *192.168.1.10*. In the *Host description* type perfSONAR1.

Test name/description
Latency Test

Test Status
 Enabled

Interface
ens32 - 192.168.2.10

[+ Advanced Parameters](#)

Test members

HOST	DESCRIPTION	IPV
------	-------------	-----

[+ Add Test Member\(s\)](#)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP 192.168.1.10	Host description perfSONAR1	IPv4 <input checked="" type="checkbox"/>	IPv6 <input type="checkbox"/>
-----------------------------	--------------------------------	--	-------------------------------

Cancel

Step 8. Click on *Add host* to save the changes

Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface

Test name/description: Latency Test Test Status: Enabled

Interface: ens32 - 192.168.2.10

[+ Advanced Parameters](#)

Test members

HOST	DESCRIPTION	IPV
------	-------------	-----

[+ Add Test Member\(s\)](#)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: 192.168.1.10 Host description: perfSONAR1 IPv4 IPv6

Add host Cancel **OK**

Step 10. Similarly, in *Hostname/IP* type the IP address of perfSONAR3 node 192.168.3.10. In the *Host description* type perfSONAR3.

Test name/description: Latency Test Test Status: Enabled

Interface: ens32 - 192.168.2.10

[+ Advanced Parameters](#)

Test members

HOST	DESCRIPTION	IPV
192.168.1.10	perfSONAR1	IPv4 <input checked="" type="checkbox"/> IPv6 <input type="checkbox"/>

[+ Add Test Member\(s\)](#)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: 192.168.3.10 Host description: perfSONAR3 IPv4 IPv6

Add host

Step 11. Click on *Add host* to save the changes.

Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface

Test name/description
Latency Test Test Status Enabled

Interface
ens32 - 192.168.2.10

[+ Advanced Parameters](#)

Test members

HOST	DESCRIPTION	IPV	
192.168.1.10	perfSONAR1	IPv4 <input checked="" type="checkbox"/> IPv6 <input type="checkbox"/>	

[+ Add Test Member\(s\)](#)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: 192.168.3.10 Host description: perfSONAR3 IPv4 IPv6

Add host

Step 12. In order to save the changes, click on *OK*.

Test members

HOST	DESCRIPTION	IPV	
192.168.1.10	perfSONAR1	IPv4 <input checked="" type="checkbox"/> IPv6 <input type="checkbox"/>	
192.168.3.10	perfSOANR3	IPv4 <input checked="" type="checkbox"/> IPv6 <input type="checkbox"/>	

[+ Add Test Member\(s\)](#)

Enter host information below to add new test members, or [browse communities](#) to add more test members.

Hostname/IP: Host Host description: Description IPv4 IPv6

Add host

[Cancel](#) **OK**

Step 13. To save the test click on *Save*.

Lab 3: Configuring Regular Tests Using perfSONAR Graphical User Interface

perfSONAR Toolkit on perfSONAR-Toolkit

View public dashboard Configuration Help

Home / Configuration / Tests

Administrative Information Host Tests

All scheduled tests

Throughput tests will be running 233% of the time

Configure tests between this host and other hosts. + Host + Test

View by: Test | Host

TEST NAME	TYPE	INTERVAL	TEST MEMBERS	ENABLED	ACTIONS
Throughput Test	Throughput - TCP	1 minute	2 hosts	☑	⚙️ 🗑️
perfSONAR Toolkit Default Traceroute Test	Traceroute	10 minutes	2 hosts	☑	⚙️ 🗑️

You've made changes that haven't been saved. One-way latency 2 hosts

Cancel Save

Step 14. Click on *View public dashboard* to get back to main page and see the throughput results.

perfSONAR Toolkit on perfSONAR-Toolkit

View public dashboard Configuration Help

Home / Configuration / Tests

Administrative Information Host Tests

All scheduled tests

Throughput tests will be running 233% of the time

This host is configured with both bandwidth and one-way latency tests. Bandwidth tests can interfere with one-way latency tests.

Configure tests between this host and other hosts. + Host + Test

View by: Test | Host

TEST NAME	TYPE	INTERVAL	TEST MEMBERS	ENABLED	ACTIONS
Latency Test	One-way latency		2 hosts	☑	⚙️ 🗑️
Throughput Test	Throughput - TCP	1 minute	2 hosts	☑	⚙️ 🗑️

Cancel Save

Step 15. After 3 minutes the data will be propagated thus, refresh the browser and scroll down until the *Test Result* section.

owamp ▾	Running	3.5.8-1.el7	861	View
pscheduler ▾	Running	1.1.5-2.el7		View
psconfig	Running	4.1.5-1.el7		View
twamp ▾	Running	3.5.8-1.el7	862	View

Test Results (2 Results) Configure tests ⚙

Search:

Results for the last...
1 week ▾

▲ SOURCE	⇅ DESTINATION	THROUGHPUT	LATENCY (MS)	LOSS
192.168.2.10 Details Traceroute	192.168.1.10	→ 3.08 Gbps ← n/a	→ 1.96 ← 2.69	→ 0 ← 0
192.168.2.10 Details Traceroute	192.168.3.10	→ 2.45 Gbps ← 1.37 Gbps	→ 13.8 ← -7.49	→ 0 ← 0

Show entries Showing 1 to 2 of 2 entries Previous **1** Next

This page is also available as [JSON](#)

The result above indicates that the throughput, latency and loss are 3.08 Gbps, 1.96ms and 0% respectively, when the source is *192.168.2.10* and the destination is *192.68.1.10*. When the source is *192.168.1.10* and the destination is *192.168.2.10*, the throughput is not available yet, the latency and loss are 2.69ms and 0% respectively. In the next row, when the source is *192.168.2.10* and the destination is *192.68.3.10*, the results of the throughput, latency and loss are 2.45 Gbps, 13.8ms and 0% respectively. On the other row, when the source is *192.168.3.10* and the destination is *192.168.2.10*, the throughput, latency and loss are 1.37 Gbps, -7.49ms and 0% respectively. Note that the results may vary.

3 Configuring R1 and R2 to emulate a Wide Area Network (WAN)

In this section, the user will modify the routers R1 and R2 in order to emulate a WAN using Network Emulator (NETEM) commands. The first modification consists in adding delay to the routers interface and, the second one consists in adding packet loss. At the end the user will visualize on the web interface how these changes affect the performance of the network.

3.1 Adding delay to interface connecting to network 192.168.2.0/24

In this section, the user will add a 50ms delay to the router R1 and router R2 using NETEM commands.

Step 1. On the topology, click on R1 and enter the username `root` and `password` as password. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 4.19.1-1.el7.elrepo.x86_64 on an x86_64

R1 login: root
Password:
```

Step 2. To identify the interface connected to the network *192.168.2.0/24*, in R1 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R1 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::1db9:17ab:218f:23a7 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:9a:bd txqueuelen 1000 (Ethernet)
    RX packets 38 bytes 2753 (2.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 32 bytes 2839 (2.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens36: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 203.0.113.2 netmask 255.255.255.0 broadcast 203.0.113.255
    inet6 fe80::c1a0:d0ce:dde9:24ce prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:8f:fa txqueuelen 1000 (Ethernet)
    RX packets 128 bytes 12751 (12.4 KiB)
    RX errors 0 dropped 4 overruns 0 frame 0
    TX packets 235 bytes 18898 (18.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.1 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::46d7:42e:b419:21a1 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e4:84 txqueuelen 1000 (Ethernet)
    RX packets 130 bytes 10161 (9.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 73 bytes 6898 (6.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R1 ~]# _
```

The output of the `ifconfig` command indicates that R1 has three interfaces. The interface *ens37* connects R1 to the network *192.168.2.0/24* and is configured with the IP address *192.168.2.1*. Thus, this interface must be used for emulation.

Step 3. In order to add a 50ms delay, in R1 CLI type the following command:

```
sudo tc qdisc add dev ens37 root netem delay 50ms
```

```
[root@R1 ~]# sudo tc qdisc add dev ens37 root netem delay 50ms
[root@R1 ~]# _
```

Step 4. On the topology, click on R2 and enter the username `root` and `password` as password. Note that the password will not be displayed while typing it.


```
CentOS Linux 7 (Core)
Kernel 4.19.1-1.el7.elrepo.x86_64 on an x86_64

R2 login: root
Password:
```

Step 5. To identify the interface connected to the network 192.168.2.0/24, in R2 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R2 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.1 netmask 255.255.255.0 broadcast 192.168.3.255
    inet6 fe80::250:56ff:feae:e5dc prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e5:dc txqueuelen 1000 (Ethernet)
    RX packets 1013392857 bytes 1488074592733 (1.3 TiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1217711915 bytes 2909379957450 (2.6 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.2 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::10a6:2962:4b7e:c21a prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:96:6a txqueuelen 1000 (Ethernet)
    RX packets 1213137503 bytes 1799533693195 (1.6 TiB)
    RX errors 0 dropped 10 overruns 0 frame 0
    TX packets 1016660699 bytes 2402663209617 (2.1 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 468 bytes 37528 (36.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 468 bytes 37528 (36.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R2 ~]#
```

The output of the `ifconfig` command indicates that R2 has two interfaces. The interface `ens37` connects R2 to the network 192.168.2.0/24. Thus, this interface must be used for emulation.

Step 6. In order to add a 50ms delay, in R2 command line type the command:

```
sudo tc qdisc add dev ens37 root netem delay 50ms
```

```
[root@R2 ~]# sudo tc qdisc add dev ens37 root netem delay 50ms
[root@R2 ~]# _
```

Step 7. After 3 minutes the data will be propagated thus, refresh the browser and scroll down until the *Test Result* section.

The screenshot displays the perfSONAR Graphical User Interface. On the left, a list of services is shown, all in a 'Running' state:

- esmond - Running (2.1.3-1.el7)
- lsregistration - Running (4.1.6-1.el7)
- owamp - Running (3.5.8-1.el7, 861)
- pscheduler - Running (1.1.6-2.el7)
- psconfig - Running (4.1.6-1.el7)
- twamp - Running (3.5.8-1.el7, 862)

Below the services list is the 'Test Results' section, which shows two results for the last 1 hour. The results are summarized in the following table:

SOURCE	DESTINATION	THROUGHPUT	LATENCY (MS)	LOSS
192.168.2.10	192.168.1.10	→ 1.48 Gbps ← 1.37 Gbps	→ 31.3 ← 20.3	→ 0 ← 0
192.168.2.10	192.168.3.10	→ 1.63 Gbps ← 1.48 Gbps	→ 31.3 ← 20.3	→ 0 ← 0

On the right side of the interface, there is a 'Host Information' panel with various system details and an 'On-demand testing tools' section containing links for Reverse ping, Reverse traceroute, Reverse tracepath, and Traceroute Visualization.

The results of the throughput, latency and loss are 1.63 Gbps, 31.3ms and 0% respectively, when the source is *192.168.2.10* and the destination is *192.68.3.10*. On the other hand, when the source is *192.168.3.10* and the destination is *192.168.2.10*, the throughput, latency and loss are 1.48 Gbps, 20.3ms and 0% respectively. Note that the results may vary.

3.2 Adding packet loss to interface connecting to network 192.168.2.0/24

In this section, the user will add a 40% packet loss to the routers R1 and R2 using Network Emulator (NETEM) command line.

Notice that 40% of loss is unrealistic for real WANs. This value is used in order to have the data propagated during this lab.

Step 1. Open R2 and enter the username `root` and password as `password`. Note that the password will not be displayed while typing it.

Step 2. To identify the interface connected to the network *192.168.2.0/24*, in R1 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R1 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::1db9:17ab:218f:23a7 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:9a:bd txqueuelen 1000 (Ethernet)
    RX packets 38 bytes 2753 (2.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 32 bytes 2839 (2.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens36: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 203.0.113.2 netmask 255.255.255.0 broadcast 203.0.113.255
    inet6 fe80::c1a0:d0ce:dde9:24ce prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:8f:fa txqueuelen 1000 (Ethernet)
    RX packets 128 bytes 12751 (12.4 KiB)
    RX errors 0 dropped 4 overruns 0 frame 0
    TX packets 235 bytes 18898 (18.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.1 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::46d7:42e:b419:21a1 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e4:84 txqueuelen 1000 (Ethernet)
    RX packets 130 bytes 10161 (9.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 73 bytes 6898 (6.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R1 ~]# _
```

The output of the `ifconfig` command indicates that R1 has also three interfaces with the same names. The interface `ens37` connects R1 to the network `192.168.2.0/24`. Thus, this interface must be used for emulation.

Step 3. In order to add 40% packet loss, in R1 command line type:

```
sudo tc qdisc change dev ens37 root netem delay 50ms loss 40%
```

```
[root@R1 ~]# sudo tc qdisc change dev ens37 root netem delay 50ms loss 40%
[root@R1 ~]# _
```

Step 4. Open R2 and enter the username `root` and password as `password`. Note that the password will not be displayed while typing it.

Step 5. To identify the interface connected to the network `192.168.2.0/24`, in R2 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R2 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.1 netmask 255.255.255.0 broadcast 192.168.3.255
    inet6 fe80::250:56ff:feae:e5dc prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e5:dc txqueuelen 1000 (Ethernet)
    RX packets 1013392857 bytes 1488074592733 (1.3 TiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1217711915 bytes 2909379957450 (2.6 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.2 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::10a6:2962:4b7e:c21a prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:96:6a txqueuelen 1000 (Ethernet)
    RX packets 1213137503 bytes 1799533693195 (1.6 TiB)
    RX errors 0 dropped 10 overruns 0 frame 0
    TX packets 1016660699 bytes 2402663209617 (2.1 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 468 bytes 37528 (36.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 468 bytes 37528 (36.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R2 ~]#
```

Notice that the interface *ens37* is connected to the network *192.168.2.0/24*.

Step 6. In order to add a 40% packet loss, type in R2 command line, type:

```
sudo tc qdisc change dev ens37 root netem delay 50ms loss 40%
```

```
[root@R2 ~]# sudo tc qdisc change dev ens37 root netem delay 50ms loss 40%
[root@R2 ~]#
```

Step 7. Go back to the Client host to see how these changes affect the performance.

The screenshot shows the perfSONAR GUI with a table of test results. The table has columns for SOURCE, DESTINATION, THROUGHPUT, LATENCY (MS), and LOSS. The selected test result is for a throughput test between source 192.168.2.10 and destination 192.168.3.10. The results are: Throughput: 1.23 Gbps (range 1.26 to 1.23), Latency: 72.4 ms (range 24.9 to 72.4), and Loss: 61.667% (range 42.389% to 61.667%).

SOURCE	DESTINATION	THROUGHPUT	LATENCY (MS)	LOSS
192.168.2.10	192.168.1.10	→ 1.62 Gbps ← 1.21 Gbps	→ 26.9 ← -17.9	→ 6.262% ← 39.643%
192.168.2.10	192.168.3.10	→ 1.23 Gbps ← 1.26 Gbps	→ 72.4 ← -24.9	→ 61.667% ← 42.389%

The results of the throughput, latency and loss are 1.23 Gbps, 72.4ms and 61.667% respectively, when the source is *192.168.2.10* and the destination is *192.68.3.10*. On the

other hand, when the source is *192.168.3.10* and the destination is *192.168.2.10*, the throughput, latency and loss are 1.26 Gbps, -24.9ms and 42.389% respectively.

This concludes Lab 3.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swany and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 4: Configuring Regular Tests Using pScheduler CLI Part I

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	4
Lab roadmap	4
1 Introduction.....	4
1.1 The pScheduler command	6
2 Latency tests.....	6
2.1 One-way ping	7
2.2 Two-way ping.....	10
2.3 Round-Trip Time (RTT)	13
3 Throughput tests	14
3.1 iPerf3.....	14
3.2 Nuttcp	15
4 Trace tests	16
4.1 Traceroute.....	17
4.2 Tracepath	18
4.3 Paris traceroute	18
References.....	20

Overview

This lab introduces the reader to pScheduler commands, and how to use the default and specific tools to run latency, throughput and trace tests. It demonstrates how to invoke the pScheduler command to properly run a measurement test using the available tools.

Objectives

By the end of this lab, the user will:

1. Understand pScheduler commands.
2. Measure latency using *owamp*, *twamp* and ping tools.
3. Run throughput tests using *iperf3* and *nuttcp* tools.
4. Use *traceroute*, *tracpath* and *paris-tracecoute* tools to identify the hops from a source to a destination.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

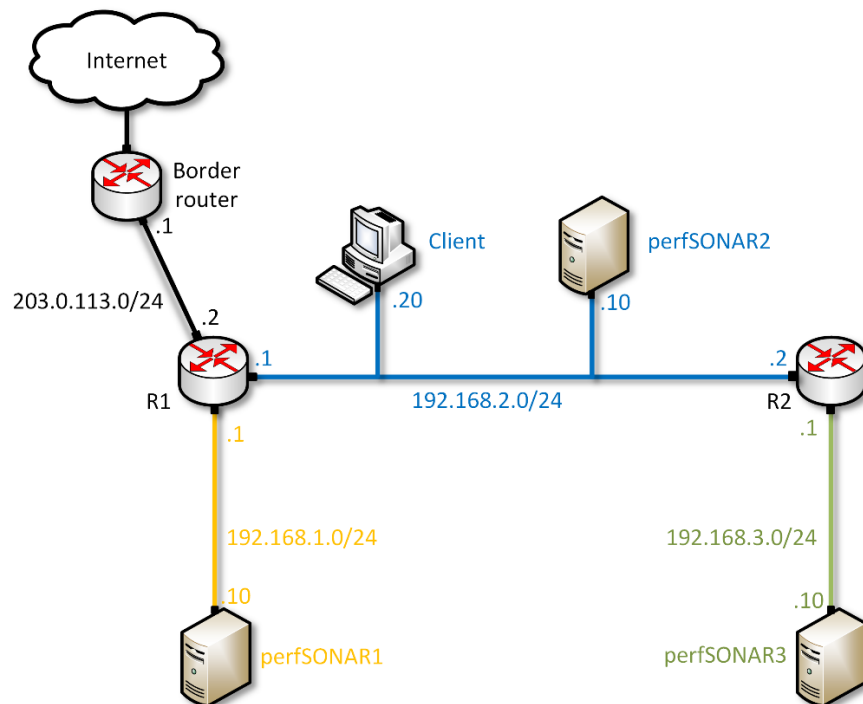


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Latency tests.
3. Section 3: Throughput tests.
4. Section 4: Trace Tests.

1 Introduction

pScheduler is responsible for managing the execution of network measurements, or more generally tasks, in perfSONAR. When the user wants to run a network measurement on perfSONAR, it is performed through pScheduler command-line. pScheduler is part of the scheduling layer, as it is shown in the figure 2. The scheduling layer is responsible for:

- Finding time-slots to run the tools while avoiding scheduling conflicts that would negatively impact results.
- Executing the tools and gathering results.
- Sending to the results to the archiving layer (if needed).

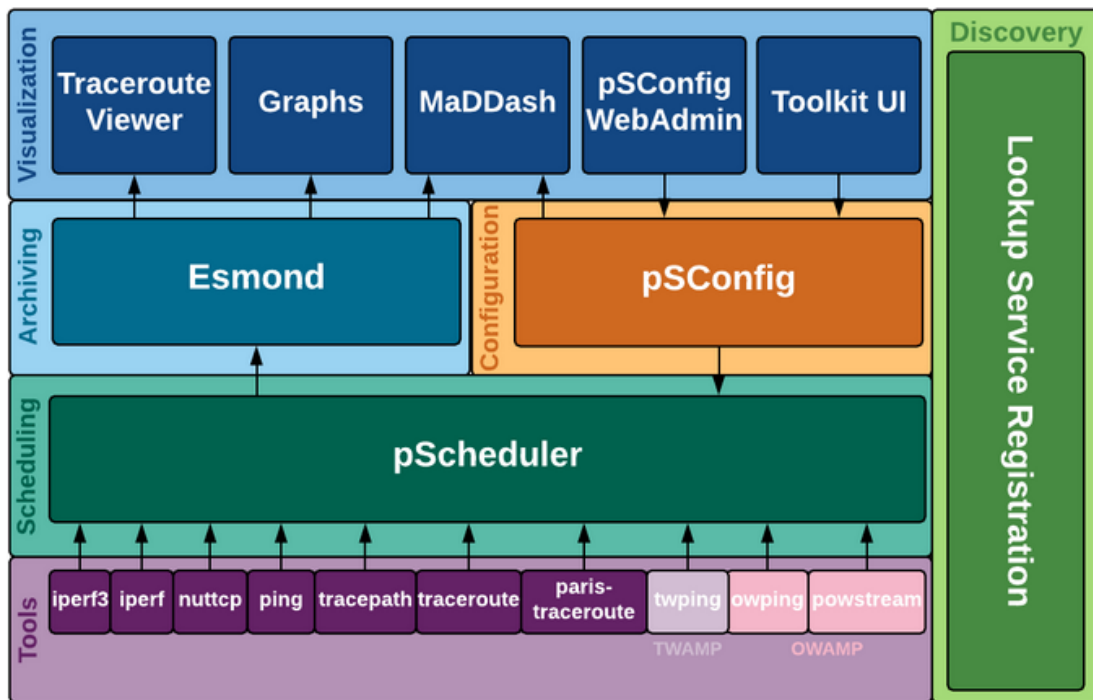


Figure 2. perfSONAR layers³.

pScheduler handles the coordination, execution and optionally storage of the task requested. Many of the tools pScheduler executes could be run independently of pScheduler. However, pScheduler provides additional features that the tool by itself does not provide. These features are listed below:

- *Measurement Integrity:* pScheduler maintains a schedule of all measurements to be run and will not allow any measurements to run simultaneously if doing so would adversely affect the result in a significant way. For instance, it will not run two throughput tests at the same time as the competition for resources could affect the results of each. In contrast, it will run latency tests in the background as the low resource consumption does not significantly affect results of parallel tests.
- *Simplified Coordination:* In addition to simplifying coordination during task execution, pScheduler will contact each end device and handle bringing up any daemons as required. It also has a plug-in architecture that allows for sending the result elsewhere, such as a long-term storage system, as well the measurement completes.
- *Access Control:* pScheduler has a limits system that allows the definition of rules about who can run what type of measurements and other rules such as how long a test can run, and which tests are allowed to run in a specific node.
- *Diagnostics:* pScheduler provides the tools to visualize the schedule. It specifies when a task ran, is running or will run. Additionally, it keeps for some amount of time information about the outcome, including whether the result was a failure or not, which can be useful for diagnosing issues within a network.

In addition to these foundational features, pScheduler allows plug-ins for new tests, tools and archivers to be written. This means that pScheduler allows extensions to

perform new type of measurements or other functions, as well as the ability to have their results sent to new types of storage and/or analysis tools.

In this lab, the user will run latency, throughput and trace tests using pScheduler command-line interface (CLI). These tests include, latency, throughput and traceroute measurements.

1.1 The pScheduler command

The user interacts with perfSONAR using `pscheduler` command. The pScheduler command is the primary way for the command-line to create new pScheduler tasks⁴. The basic syntax is as follows:

```
pscheduler command [args] (1)
```

`pscheduler`: command used to interact with perfSONAR.

`command`: describes the type of test that will be performed. These commands could be task commands or administrative and diagnosis commands, and each command has its own lists of arguments `args`. The task commands are listed as follows:

- `task`: give pScheduler a task that consists of making one or more measurements.
- `result`: fetch and display the results of a single, previously-concluded run by its URL.
- `watch`: attach to a task identified by URL and show run results as they become available.
- `cancel`: stop any future runs of a task.

The following commands are for diagnosis and administrative:

- `ping`: determine if pScheduler is running on a host.
- `clock`: check and compare the clocks on pScheduler hosts.
- `debug`: Enable debugging on the internal part of pScheduler.
- `diags`: Produce a diagnostic dump for the perfSONAR team to use in resolving problems.

For more information about pScheduler tasks, diagnosis, and administrative commands, the user can get access to the help by typing on the perfSONAR command-line:

```
pscheduler -help
```

To get more details about a specific command, using the format of the command (1) type:

```
pscheduler [command] --help
```

2 Latency tests

In this section, the user will run latency measurement tests using pScheduler tools. pScheduler uses One-Way Ping (OWPING), Two-Way Ping (TWPING) and Round-Trip Time (RTT) to measure the latency as shown in the tools layer in figure 2. First, the user will run a latency test using the default configuration then, the user will specify a tool to run a latency test.

2.1 One-way ping

In this part, the user will run a latency test between perfSONAR1 (192.168.1.10) and perfSONAR2 (192.168.2.10). The default tool used to perform this measurement is one-way ping (*owping*), as shown in figure 2. The user interacts with pScheduler using command-line interface (CLI).

Step 1. On the topology, click on perfSONAR1 then, enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it. Proceed similarly with perfSONAR2 and perfSONAR3.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar1 login: admin
Password:
Last login: Wed Jan 30 15:14:47 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar1 ~]$\
```

Step 2. In perfSONAR1 command line, follow command format (1) and type:

```
pscheduler task latency --source 192.168.1.10 --dest 192.168.2.10
```

- `pscheduler`: command to interact with perfSONAR.
- `task`: pScheduler command to specify a measurement test.
- `latency`: test type.
- `--source`: specify where the test should originate, in this case it is perfSONAR1 node (192.168.1.10).
- `--dest`: destination node, in this case it is the perfSONAR2 node (192.168.2.10).

```
ladmin@perfsonar1 ~1$ pscheduler task latency --source 192.168.1.10 --dest 192.168.2.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/0ecd9b95-a0f2-4f7f-9ac3-907058d6f79f
Running with tool 'owping'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/0ecd9b95-a0f2-4f7f-9ac3-907058d6f79f/runs/9d621b92-3bc1-4f3c-9
854-23c1a19432cf
Starts 2019-05-07T18:40:46Z (~7 seconds)
Ends 2019-05-07T18:41:07Z (~20 seconds)
Waiting for result...
```

The default tool used by pScheduler to run the default test is one-way ping (*owping*) tool. The task is scheduled, and the results are shown below. There are three sections in the report:

- *Packet Statistics*: It shows a summary of the number of sent and received packets, as well as the number packet lost, duplicated and reordered. The Packet Statistics of the last test is shown in the table 2.
- *One-way Latency Statistics*: It summarizes the One-way Latency Statistics as shown in the table 3 and a Histogram of the delay values.
- *TTL Statistics*: It shows the time-to-live statistics. The results are shown in the table 4, where the values do not vary.

In order to navigate through the result summary, press `Shift+Page Up` to scroll up `Shift+Page Down` to scroll down.

```

Packet Statistics
-----
Packets Sent ..... 100 packets
Packets Received ..... 100 packets
Packets Lost ..... 0 packets
Packets Duplicated ... 0 packets
Packets Reordered ... 0 packets

One-way Latency Statistics
-----
Delay Median ..... 1.33 ms
Delay Minimum ..... 1.26 ms
Delay Maximum ..... 2.03 ms
Delay Mean ..... 1.34 ms
Delay Mode ..... 1.36 ms
Delay 25th Percentile ... 1.30 ms
Delay 75th Percentile ... 1.36 ms
Delay 95th Percentile ... 1.43 ms
Max Clock Error ..... 6.75 ms
Common Jitter Measurements:
  P95 - P50 ..... 0.10 ms
  P75 - P25 ..... 0.06 ms
  Variance ..... 0.01 ms
  Std Deviation .... 0.09 ms
Histogram:
  1.26 ms: 4 packets
  1.27 ms: 7 packets
  1.28 ms: 7 packets
  1.29 ms: 2 packets
  1.30 ms: 8 packets
  1.31 ms: 9 packets
  1.32 ms: 8 packets
  1.33 ms: 8 packets
  1.34 ms: 6 packets
  1.35 ms: 9 packets
  1.36 ms: 12 packets
  1.37 ms: 7 packets
  1.38 ms: 5 packets
  1.40 ms: 1 packets
  1.41 ms: 2 packets
  1.43 ms: 1 packets
  1.44 ms: 1 packets
  1.52 ms: 1 packets
  1.61 ms: 1 packets
  2.03 ms: 1 packets

TTL Statistics
-----
TTL Median ..... 254.00
TTL Minimum ..... 254.00
TTL Maximum ..... 254.00
TTL Mean ..... 254.00
TTL Mode ..... 254.00
TTL 25th Percentile ... 254.00
TTL 75th Percentile ... 254.00
TTL 95th Percentile ... 254.00
Histogram:
  254: 100 packets

No further runs scheduled.
    
```

Table 2. Packet Statistics.

Packet Sent	100 packets
Packet Received	100 packets
Packet Lost	0 packets
Packet Duplicated	0 packets
Packet Reordered	0 packets

Table 3. One-way Latency Statistics.

Delay Median	1.33ms
Delay Minimum	1.26ms
Delay Maximum	2.03ms
Delay Mean	1.34ms

Delay Mode	1.36ms
Delay 25th Percentile	1.30ms
Delay 75th Percentile	1.36ms
Delay 95th Percentile	1.43ms
Max Clock Error	6.75ms

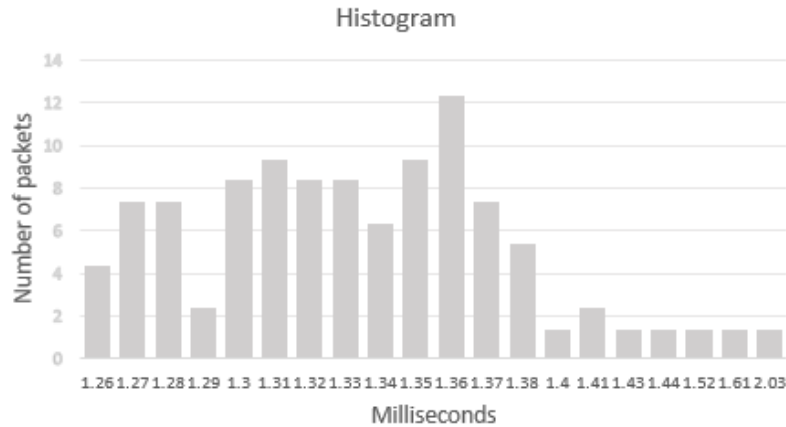


Figure 3. One-way Latency Statistics Histogram.

Table 4. TTL Statistics.

TTL Median	254
TTL Minimum	254
TTL Maximum	254
TTL Mean	254
TTL Mode	254
TTL 25th Percentile	254
TTL 75th Percentile	254
TTL 95th Percentile	254

2.2 Two-way ping

In the following steps, the user will specify a tool to run a two-way ping (*twping*) test between perfSONAR1 (192.168.1.10) and perfSONAR2 (192.168.2.10) using pscheduler command-line.

Step 1. In perfSONAR1 command line, follow command format (1) and type:

```
pscheduler task --tool twping latency --source 192.168.1.10 --dest 192.168.2.10
```

- `pscheduler`: command to interact with perfSONAR.
- `task`: pScheduler command to specify a measurement test.
- `--tool`: command to specify the tool.
- `twping`: tool for two-way ping measurement.
- `latency`: test type.
- `--source`: specify where the test should originate, in this case it is perfSONAR1 node (192.168.1.10).

- `--dest`: the destination node, in this case it is the perfSONAR2 node (192.168.2.10).

```
admin@perfsonar1 ~1$ pschedler task --tool twping latency --source 192.168.1.10 --dest 192.168.2.10
-bash: pschedler: command not found
admin@perfsonar1 ~1$ pscheduler task --tool twping latency --source 192.168.1.10 --dest 192.168.2.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/4e5004a6-37b4-4550-b880-4624b51c6ec1
Running with tool 'twping'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/4e5004a6-37b4-4550-b880-4624b51c6ec1/runs/f4f8d848-47e3-4105-a75a-ab6d017079a2
Starts 2019-05-08T14:14:04Z (~8 seconds)
Ends 2019-05-08T14:14:25Z (~20 seconds)
Waiting for result...
```

In this case, the user specifies two-way ping (*twping*) as the tool to run the measurement test. The task is scheduled, and the results are shown below. The report format is like the latency report.

In order to navigate through the result summary, press `Shift+Page Up` to scroll up `Shift+Page Down` to scroll down.


```

Packet Statistics
-----
Packets Sent ..... 100 packets
Packets Received .... 100 packets
Packets Lost ..... 0 packets
Packets Duplicated ... 0 packets
Packets Reordered .... 0 packets

One-way Latency Statistics
-----
Delay Median ..... 1.99 ms
Delay Minimum ..... 1.92 ms
Delay Maximum ..... 2.21 ms
Delay Mean ..... 2.00 ms
Delay Mode ..... 1.99 ms
Delay 25th Percentile ... 1.97 ms
Delay 75th Percentile ... 2.01 ms
Delay 95th Percentile ... 2.07 ms
Max Clock Error ..... 8.67 ms
Common Jitter Measurements:
  P95 - P50 ..... 0.08 ms
  P75 - P25 ..... 0.04 ms
  Variance ..... 0.00 ms
  Std Deviation .... 0.05 ms

Histogram:
  1.92 ms: 1 packets
  1.93 ms: 3 packets
  1.94 ms: 3 packets
  1.95 ms: 7 packets
  1.96 ms: 10 packets
  1.97 ms: 8 packets
  1.98 ms: 11 packets
  1.99 ms: 16 packets
  2.00 ms: 14 packets
  2.01 ms: 8 packets
  2.02 ms: 4 packets
  2.03 ms: 2 packets
  2.05 ms: 2 packets
  2.06 ms: 4 packets
  2.07 ms: 3 packets
  2.13 ms: 2 packets
  2.14 ms: 1 packets
  2.21 ms: 1 packets

TTL Statistics
-----
TTL Median ..... 254.00
TTL Minimum ..... 254.00
TTL Maximum ..... 254.00
TTL Mean ..... 254.00
TTL Mode ..... 254.00
TTL 25th Percentile ... 254.00
TTL 75th Percentile ... 254.00
TTL 95th Percentile ... 254.00
Histogram:
  254: 100 packets

No further runs scheduled.
    
```

Table 5. Packet Statistics.

Packet Sent	100 packets
Packet Received	100 packets
Packet Lost	0 packets
Packet Duplicated	0 packets
Packet Reordered	0 packets

Table 6. One-way Latency Statistics.

Delay Median	1.99ms
Delay Minimum	1.92ms
Delay Maximum	2.21ms
Delay Mean	2.00ms
Delay Mode	1.99ms

Delay 25th Percentile	1.97ms
Delay 75th Percentile	2.01ms
Delay 95th Percentile	2.07ms
Max Clock Error	8.67ms

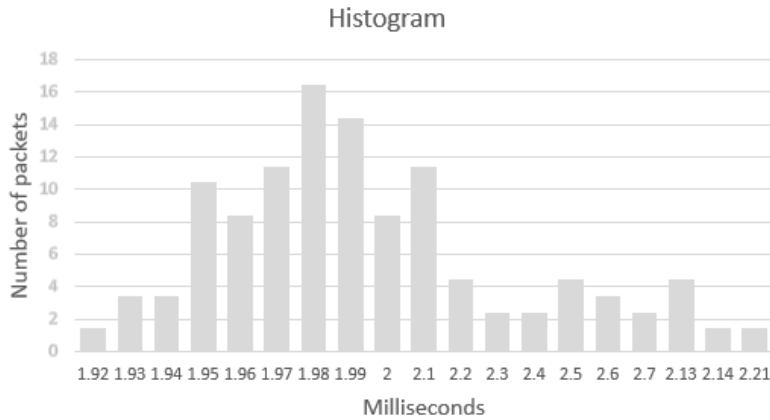


Figure 4. One-way Latency Statistics Histogram.

Table 7. TTL Statistics.

TTL Median	254
TTL Minimum	254
TTL Maximum	254
TTL Mean	254
TTL Mode	254
TTL 25th Percentile	254
TTL 75th Percentile	254
TTL 95th Percentile	254

2.3 Round-Trip Time (RTT)

In this part, the user will run a Round-Trip Time (*RTT*) test is between perfSONAR1 (192.168.1.10) and perfSONAR2 (192.168.2.10) of the given topology, see figure 1. First, the user is going to type a pScheduler command to run a test in perfSONAR1 command-line, then the user will repeat the same test in perfSONAR3 (192.168.3.10) command-line.

Step 1. In perfSONAR1 command line, follow command format (1) and type:

```
pscheduler task rtt --source 192.168.1.10 --dest 192.168.2.10
```

- `pscheduler`: command to interact with perfSONAR.
- `task`: pScheduler command to specify a measurement test.
- `rtt`: test type.
- `--source`: it specifies where the test should originate, in this case it is perfSONAR1 node (192.168.1.10).
- `--dest`: destination node, in this case it is the perfSONAR2 node (192.168.2.10).

```

[admin@perfsonar1 ~]# pscheduler task rtt --source 192.168.1.10 --dest 192.168.2.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/285cb7b9-97d2-4a19-b2fe-4d394d826502
Running with tool 'ping'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/285cb7b9-97d2-4a19-b2fe-4d394d826502/runs/0066a9cc-6865-419b-abf3-57186f1dd012
Starts 2019-01-30T21:58:56Z (~8 seconds)
Ends 2019-01-30T21:59:07Z (~10 seconds)
Waiting for result...

1      192.168.2.10  64 Bytes  TTL 63  RTT  0.3510 ms
2      192.168.2.10  64 Bytes  TTL 63  RTT  0.3580 ms
3      192.168.2.10  64 Bytes  TTL 63  RTT  0.3610 ms
4      192.168.2.10  64 Bytes  TTL 63  RTT  0.3760 ms
5      192.168.2.10  64 Bytes  TTL 63  RTT  0.2900 ms

0% Packet Loss  RTT Min/Mean/Max/StdDev = 0.290000/0.347000/0.376000/0.032000 ms

No further runs scheduled.
[admin@perfsonar1 ~]#

```

The result above indicates that all five packets were received successfully by perfSONAR2 node (192.168.2.10) (0% packet loss) and that the minimum, mean, maximum, and standard deviation of the Round-Trip Time (RTT) were 0.290, 0.347, 0.376 and 0.032 milliseconds respectively.

3 Throughput tests

In this section, the user will run throughput measurement tests using pScheduler tools. These tools are iperf, iperf3 and nuttcp as shown in the tools layer in the figure 2. First, the user will run a throughput test using the default configuration then, the user will specify a tool to run a latency test.

3.1 iPerf3

The following throughput test is between perfSONAR1 node (192.168.1.10) and perfSONAR2 node (192.168.2.10). The tool used to run the default test is *iperf3*.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10
```

- `pscheduler`: command to interact with perfSONAR.
- `task`: pScheduler command.
- `throughput`: test type.
- `--source`: specify where the test should originate, in this case it is perfSONAR1 node (192.168.1.10).
- `--dest`: destination node, in this case it is the perfSONAR2 node (192.168.2.10).

```

[admin@perfsonar1 ~]$ pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/05d8afad-0096-4284-ad20-4de6a3cb8674
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/05d8afad-0096-4284-ad20-4de6a3cb8674/runs/69129980-9fb1-4d95-a
130-f722206f57d1
Starts 2019-01-30T22:41:34Z (~8 seconds)
Ends 2019-01-30T22:41:53Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput      Retransmits      Current Window
0.0 - 1.0    8.59 Gbps       791              918.03 KBytes
1.0 - 2.0    7.71 Gbps       231              829.70 KBytes
2.0 - 3.0    8.31 Gbps       0                974.50 KBytes
3.0 - 4.0    8.16 Gbps       227              802.19 KBytes
4.0 - 5.0    8.23 Gbps       0                932.51 KBytes
5.0 - 6.0    8.14 Gbps       289              845.63 KBytes
6.0 - 7.0    8.33 Gbps       0                999.12 KBytes
7.0 - 8.0    8.04 Gbps       251              822.46 KBytes
8.0 - 9.0    8.41 Gbps       0                935.41 KBytes
9.0 - 10.0   8.14 Gbps       281              809.43 KBytes

Summary
Interval      Throughput      Retransmits
0.0 - 10.0    8.20 Gbps       2070

No further runs scheduled.
[admin@perfsonar1 ~]$

```

Shortly after starting the test submission, the user will see that the tool used to run the test is iperf3. The results above list the throughput every second (*Interval*), the number of retransmissions (Retransmits) and current windows size. At the end, it is summarized the time interval when the test took place, in this case from 0 seconds to 10 seconds, the throughput is 8.20 Gbps and the number of retransmissions is 2070.

3.2 Nuttcp

The following throughput test is between perfSONAR1 node (*192.168.1.10*) and perfSONAR2 node (*192.168.2.10*). The tool used to this test is nuttcp.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task --tool nuttcp throughput --source 192.168.1.10 --dest
192.168.2.10 -i1
```

- `pscheduler`: command to interact with perfSONAR.
- `task`: pScheduler command.
- `--tool`: command to specify the tool.
- `nuttcp`: tool used to run the test.
- `throughput`: test type.
- `--source`: specify where the test should originate, in this case it is perfSONAR1 node (*192.168.1.10*).
- `--dest`: destination node, in this case it is the perfSONAR2 node (*192.168.2.10*).
- `i1`: indicates the interval is 1 second.

```

[admin@perfsonar1 ~]# pscheduler task --tool nuttcp throughput --source 192.168.1.10 --dest 192.168.2.10 -i 1
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/b9e3ef72-375f-4e8a-80ba-f4acfab92f8
Running with tool 'nuttcp'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/b9e3ef72-375f-4e8a-80ba-f4acfab92f8/runs/f88fc6ed-aef6-4b90-8d0d-632e0ba37d50
Starts 2019-06-06T00:11:51Z (~7 seconds)
Ends 2019-06-06T00:12:07Z (~15 seconds)
Waiting for result...

Run did not complete: Failed

Diagnostics from 192.168.1.10:
/usr/bin/nuttcp -p 5101 -P 5001 -4 -T 10 -i 1 192.168.2.10

763.0625 MB / 1.00 sec = 6400.6354 Mbps 2214 retrans 535 KB-cwnd
744.6875 MB / 1.00 sec = 6247.2351 Mbps 151 retrans 800 KB-cwnd
746.3125 MB / 1.00 sec = 6260.4228 Mbps 0 retrans 929 KB-cwnd
713.2500 MB / 1.00 sec = 5983.2405 Mbps 322 retrans 687 KB-cwnd
730.6250 MB / 1.00 sec = 6128.9267 Mbps 0 retrans 862 KB-cwnd
703.7500 MB / 1.00 sec = 5903.0697 Mbps 274 retrans 694 KB-cwnd
720.5000 MB / 1.00 sec = 6044.3305 Mbps 0 retrans 883 KB-cwnd
733.6875 MB / 1.00 sec = 6154.2353 Mbps 277 retrans 701 KB-cwnd
739.6250 MB / 1.00 sec = 6204.2505 Mbps 0 retrans 938 KB-cwnd
740.9375 MB / 1.00 sec = 6214.5891 Mbps 344 retrans 779 KB-cwnd

7357.4375 MB / 10.03 sec = 6155.7772 Mbps 20 %TX 23 %RX 3582 retrans 706 KB-cwnd 0.37 msRTT

Error from 192.168.1.10:
No error.

Diagnostics from 192.168.2.10:
/usr/bin/nuttcp -S -1 --nofork -p 5101 -P 5001 -4

Error from 192.168.2.10:
No error.

No further runs scheduled.
[admin@perfsonar1 ~]# s_

```

The results above indicate that the test did not failed. After that it is shown the nuttcp command used to run the task from local host or perfSONAR1 node (192.168.1.10). The summarized data indicates that 8133.3174 MB were transferred in 10.03 seconds. This is equivalent to 6800.8015 Mbps. The results also show the CPU usage, which in this case is 20% for the transmitter (TX) and 24% for the receiver (RX). The number of retransmissions is 1052, the congestion windows size is 951 KB, and the Round-Trip Time (RTT) is 0.31ms. In addition, it is shown the nuttcp command used to run the server in perfSONAR2 node (192.168.2.10). Finally, there is a report indicating that there is not any error from perfSONAR2 node.

In order to navigate through the result summary, press `Shift+Page Up` to scroll up `Shift+Page Down` to scroll down.

4 Trace tests

In this section, the user will run trace measurement tests using pScheduler tools. These tools are *traceroute*, *tracpath* and *paris-traceroute* as shown in the tools layer in the figure 2. First, the user will run a trace test using the default configuration then, then the user will specify a tool to run a latency test.

4.1 Traceroute

Traceroute measures the path that a packet took as it traveled around the Internet to the website. It also displays the response times that occurred at each stop along the route. If there is a connection problem or latency connecting to a site, it will show up in these response times. The user will be able to identify which of the hops along the route may cause a problem³.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task trace --source 192.168.1.10 --dest 192.168.3.10
```

```

[admin@perfsonar1 ~]# pscheduler task trace --source 192.168.1.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/e7daf2bb-748f-426b-841b-5342e9eb693e
Running with tool 'traceroute'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/e7daf2bb-748f-426b-841b-5342e9eb693e/runs/e88aa588-1c7a-484f-b8b5-527c31399f67
Starts 2019-06-06T00:19:03Z (~8 seconds)
Ends 2019-06-06T00:19:11Z (~7 seconds)
Waiting for result...

1      gateway (192.168.1.1) 0.2 ms
2      192.168.2.2 0.9 ms
3      192.168.3.10 0.9 ms

No further runs scheduled.
[admin@perfsonar1 ~]#
    
```

Shortly after submitting the test, the default tool used to run the test is *traceroute*. In the figure above, there are several rows divided into columns on the report. Each row represents a hop along the route. In each hop, the packet gets its next set of directions. Each row is divided into five columns. A sample row is shown below:

HOP NUMBER	IP ADDRESS	RTT
1	192.168.1.1	0.2 ms
2	192.168.2.2	0.9 ms
3	192.168.3.10	0.9 ms

- **HOP NUMBER:** It represents the number of the hop along the route. In this case, it takes two hops to reach the destination.
- **IP ADDRESS:** The second column has the IP address of the destination; the previous hop has the IP address of the router. If it is available, the domain name will also be listed.

- *RTT*: The next column displays the Round-Trip Time (RTT) for the packet to reach that point and return to the source host. This measure is listed in milliseconds.

4.2 Tracepath

Tracepath traces a path from the source to destination, discovering the Maximum Transmission Unit (MTU) along this path. It uses UDP port or some random port. The difference from *traceroute* is that this tool includes less options and the user is not required to be a superuser to run the tests. pScheduler allows the selection of the tool *tracepath* using the `--tool` command.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task --tool tracepath trace --source 192.168.1.10 --dest 192.168.3.10
```

```

admin@perfsonar1 ~1$ pscheduler task --tool tracepath trace --source 192.168.1.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/7112f20e-448f-468f-96ad-f0086e74fa64
Running with tool 'tracepath'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/7112f20e-448f-468f-96ad-f0086e74fa64/runs/6a07f051-738d-4b3b-8ad0-e956f67e73fd
Starts 2019-05-09T10:23:50Z (~8 seconds)
Ends 2019-05-09T10:25:31Z (~100 seconds)
Waiting for result...

1 gateway (192.168.1.1) 0.148 ms mtu 1500 bytes
2 192.168.2.2 0.389 ms mtu 1500 bytes
3 192.168.3.10 0.461 ms mtu 1500 bytes host-unreachable

No further runs scheduled.
admin@perfsonar1 ~1$ _
    
```

The first column shows the hop number, the second column shows the IP address or Domain name. The third column shows the Round-Trip Time (RTT) for the packet to reach that point and return to the source host. The last column shows the Maximum Transmission Unit (MTU) size.

HOP NUMBER	IP ADDRESS	RTT	MTU
1	192.168.1.1	0.148 ms	1500 bytes
2	192.168.2.2	0.389 ms	1500 bytes
3	192.168.3.10	1.4 ms	1500 bytes

4.3 Paris traceroute

Paris traceroute is a new version of the *traceroute* network diagnosis tool. It addresses problems caused by load balancers with the initial traceroute implementation. By controlling the flow identifier of the probes, it can follow accurate paths in networks with load balancers. It is also able to find all the load balanced paths to the destination. Finally, it complements its output with information extracted from the received packets, allowing

a more precise analysis of the discovered paths. Paris traceroute, by controlling packet header contents, obtains a more precise picture of the actual routes that packets follow⁴. The user can select *paris-traceroute* tool using pScheduler `--tool` command.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task --tool paris-traceroute trace --source 192.168.1.10 --dest 192.168.3.10
```

```
[admin@perfsonar1 ~]# pscheduler task --tool paris-traceroute trace --source 192.168.1.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/00f5d73d-162d-4865-b8a3-726c87d4abad
Running with tool 'paris-traceroute'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/00f5d73d-162d-4865-b8a3-726c87d4abad/runs/5c89c788-47b1-40fa-a251-ee540feb6679
Starts 2019-05-09T10:33:30Z (~8 seconds)
Ends 2019-05-09T10:33:53Z (~22 seconds)
Waiting for result...

1 gateway (192.168.1.1) 0.241 ms
2 192.168.2.2 0.343 ms
3 192.168.3.10 4.515 ms

No further runs scheduled.
[admin@perfsonar1 ~]#
```

Notice that it is not possible to prove the idea of Paris traceroute in the current lab topology.

After the task is submitted, the user can see that the tool selected is *paris-traceroute*. The results of Paris-traceroute test are interpreted in the same way of traceroute test. In the figure above, there are several rows divided into columns on the report. These results are reordered in the table below. Each row represents a hop along the route. In each hop, the packet gets its next set of directions. Each row is divided into five columns. A sample row is shown below:

HOP NUMBER	IP ADDRESS	RTT
1	192.168.1.1	0.241 ms
2	192.168.2.2	0.343 ms
3	192.168.3.10	4.515 ms

- **HOP NUMBER:** It represents the number of the hop along the route. In this case, it takes two hops to reach out the destination.
- **IP ADDRESS:** The second column has the IP address of the destination; the previous hop has the IP address of the router. If it is available, the domain name will also be listed.
- **RTT Columns:** The next three columns display the Round-Trip Time (RTT) for the packet to reach that point and return to the source host. This measure is listed in milliseconds. There are three columns, because the traceroute sends three separate signal packets. This is to display consistency, or a lack thereof, in the route.

This concludes Lab 4.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swany and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 5: Configuring Regular Tests Using pScheduler CLI Part II

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	4
Lab roadmap.....	4
1 Introduction.....	4
1.1 The pScheduler command	6
2 Running tasks from other perfSONAR nodes	6
3 Repeating tasks.....	8
4 Exporting and importing tasks to JSON	9
5 Viewing the schedule.....	11
5.1 pScheduler monitor	11
5.2 pScheduler schedule	12
6 Canceling tasks	13
References.....	16

Overview

This lab continues the description of pScheduler commands, and how to use it to run measurement tests between perfSONAR nodes. This lab is focused on running a pScheduler task from other nodes, repeating, exporting and importing tasks. In addition, the tools to visualize the schedule are presented. Finally, the user will learn about the procedure to cancel a task.

Objectives

By the end of this lab, the user will:

1. Understand pScheduler commands.
2. Run tasks from other perfSONAR nodes.
3. Repeat a specific task.
4. Export and import pScheduler tasks.
5. Use the visualization tools.
6. Cancel a specific task.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

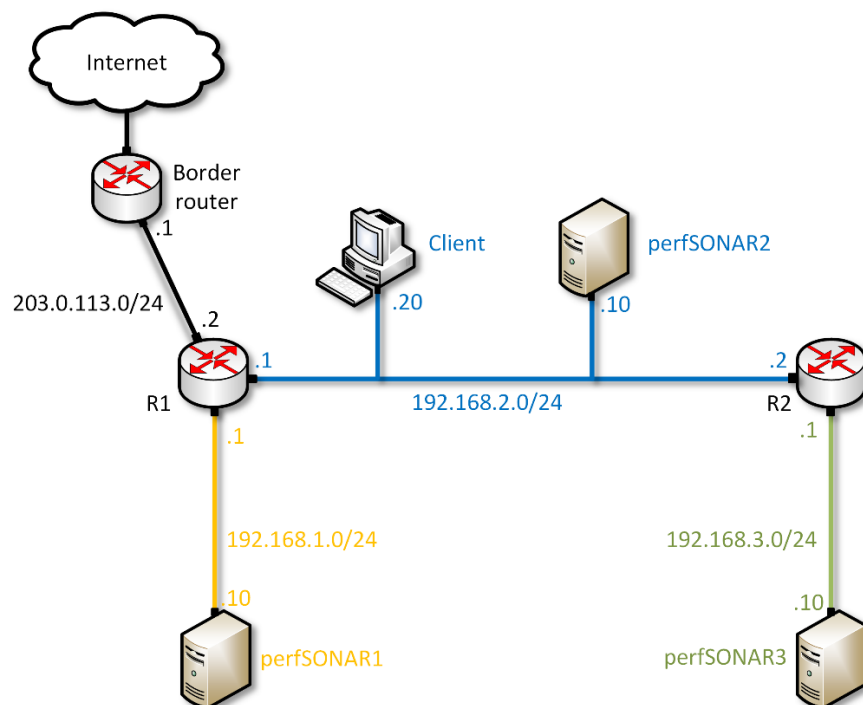


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

The lab includes the following tasks:

1. Section 1: Introduction.
2. Section 2: Running tasks from other perfSONAR nodes.
3. Section 3: Repeating tasks.
4. Section 4: Exporting and importing tasks.
5. Section 5: Viewing the schedule.
6. Section 6: Cancelling tasks.

1 Introduction

pScheduler is responsible for managing the execution of network measurements, or more generally tasks, in perfSONAR. When the user wants to run a network measurement on perfSONAR, it is performed through pScheduler command-line. pScheduler is part of the scheduling layer, as it is shown in the figure 2. The scheduling layer is responsible for:

- Finding timeslots to run the tools while avoiding scheduling conflicts that would negatively impact results.
- Executing the tools and gathering results.
- Sending to the results to the archiving layer (if needed).

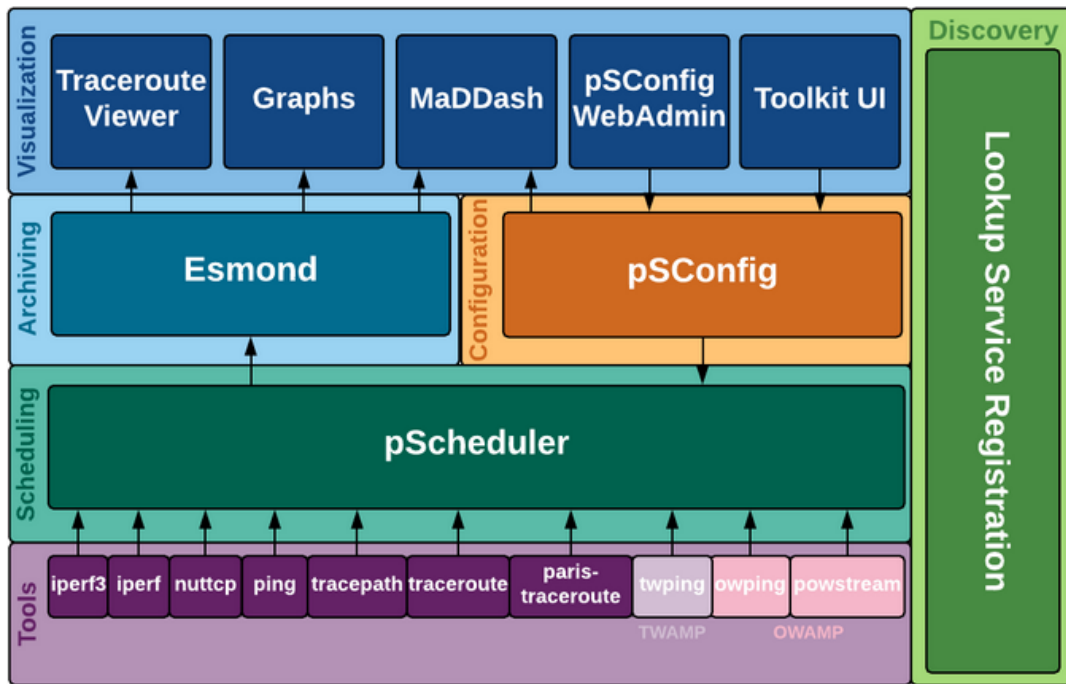


Figure 2. perfSONAR layers³.

pScheduler handles the coordination, execution and optionally storage of the task requested. Many of the tools pScheduler executes could be run independently of pScheduler. However, pScheduler provides additional features that the tools by itself does not provide. These features are listed below:

- *Measurement Integrity:* pScheduler maintains a schedule of all measurements to be run and will not allow any measurements to run simultaneously, if doing so, it would adversely affect the result in a significant way. For instance, it will not run two throughput tests at the same time as the competition for resources could affect the results of each. In contrast, it will run latency tests in the background as the low resource consumption does not significantly affect results of parallel tests.
- *Simplified Coordination:* In addition to simplify coordination during task execution, pScheduler will contact each end device and handle bringing up any daemons as required. It also has a plug-in architecture that allows to send the result elsewhere, such as a long-term storage system, as well the measurement completes.
- *Access Control:* pScheduler has a limits system that allows the definition of rules about who can run what type of measurements and other rules as how long a test can run, and which tests can run in a specific node.
- *Diagnostics:* pScheduler provides the tools to visualize the schedule. It specifies when a task ran, runs or will run. Additionally, it keeps for some amount of time information about the outcome, including whether the result was a failure or not, which can be useful for diagnosing issues with in a network.

In addition to these foundational features, pScheduler allows plug-ins for new tests, tools and archivers to be written. This means that pScheduler allows extensions to perform new type of measurements or other functions as well as to have their results sent to new types of storage and/or analysis tools.

1.1 The pScheduler command

The user interacts with perfSONAR using `pscheduler` command. The pScheduler command is the primary way from the command-line to create new pScheduler tasks⁴. The basic syntax is as follows:

```
pscheduler command [args] (1)
```

`pscheduler`: command used to interact with perfSONAR.

`command`: it describes the type of test that will be performed, these commands could be task commands or administrative and diagnosis commands, each command has its lists of arguments `args`. The task commands are listed as follows:

- `task`: give pScheduler a task that consists of making one or more measurements.
- `result`: fetch and display the results of a single, previously-concluded run by its URL.
- `watch`: attach to a task identified by URL and show run results as they become available.
- `cancel`: stop any future runs of a task.
-

The following commands are for diagnosis and administrative:

- `ping`: determine if pScheduler is running on a host.
- `clock`: checks and compare the clocks on pScheduler hosts.
- `debug`: Enable debugging on the internal part of pScheduler.
- `diags`: Produce a diagnostic dump for the perfSONAR team to use in resolving problems.

For more information about pScheduler tasks, diagnosis and administrative commands, the user get access to the help typing on the perfSONAR command-line:

```
pscheduler --help
```

To get more details about a specific command, using the format of the command (1) type:

```
pscheduler [command] --help
```

2 Running tasks from other perfSONAR nodes

pScheduler determines where to submit a task based on the test parameters. Where a task needs to be submitted is called the lead participant. For many tests run by perfSONAR, a `--source` switch which specifies where the test should originate and is also the lead participant. In this section the user will run a throughput test using pScheduler commands.

This test will be submitted by perfSONAR1 node (192.168.1.10), however, the tests will run between perfSONAR2 (192.168.2.10) and perfSONAR3 (192.168.3.10) nodes.

Step 1. On the topology, click on perfSONAR1 then, enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it. Proceed similarly with perfSONAR2 and perfSONAR3 nodes.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar1 login: admin
Password:
Last login: Wed Jan 30 15:14:47 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

    /usr/lib/perfsonar/scripts/mptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar1 ~]$\
```

Step 2. In perfSONAR1 command line, follow command format (1) and type:

```
pscheduler task throughput --source 192.168.2.10 --dest 192.168.3.10
```

- `pscheduler`: is the command to interact with perfSONAR.
- `task`: is a pScheduler command to specify a measurement test.
- `throughput`: specifies the test.
- `--source`: is to specify where the test should originate, in this case it is perfSONAR1 node (192.168.1.10)
- `--dest`: is the destination node, in this case is the perfSONAR3 node (192.168.3.10).


```

[admin@perfsonar1 ~]# pscheduler task throughput --source 192.168.2.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.2.10/pscheduler/tasks/eb890024-d060-4d70-87fa-58549c35e1d9
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.2.10/pscheduler/tasks/eb890024-d060-4d70-87fa-58549c35e1d9/runs/66fbdc76-25f5-4abd-a48e-94fe92c3c6ec
Starts 2019-05-09T16:08:22Z (~8 seconds)
Ends 2019-05-09T16:08:41Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput      Retransmits      Current Window
0.0 - 1.0    7.25 Gbps       0                 7.58 MBytes
1.0 - 2.0    6.75 Gbps       619               1.89 MBytes
2.0 - 3.0    6.68 Gbps       423               948.44 KBytes
3.0 - 4.0    6.51 Gbps       86                648.70 KBytes
4.0 - 5.0    6.48 Gbps       68                726.90 KBytes
5.0 - 6.0    6.58 Gbps       169               647.26 KBytes
6.0 - 7.0    6.56 Gbps       0                 757.30 KBytes
7.0 - 8.0    6.64 Gbps       34                634.22 KBytes
8.0 - 9.0    6.60 Gbps       107               728.34 KBytes
9.0 - 10.0   6.58 Gbps       29                629.88 KBytes

Summary
Interval      Throughput      Retransmits
0.0 - 10.0    6.66 Gbps       1535

No further runs scheduled.
[admin@perfsonar1 ~]#

```

Shortly after starting the test submission, the user will see that the tool used to run the test is iperf3. The results above, lists the throughput every second (*Interval*), the number of retransmissions (Retransmits) and current windows size. At the end, it is summarized the time interval when the test took place, in this case from 0 seconds to 10 seconds, the throughput is 6.66 Gbps and the number of retransmissions is 1535.

In this example, the command above is run on perfSONAR1 node (192.168.1.10), then the node will submit the task to pefSONAR2 node (192.168.2.10) and the test will be run between perfSOANR2(192.168.2.10) and perfSONAR3 (192.168.3.10).

3 Repeating tasks

A task can be configured to run periodically. In this section, it is shown step by step how to repeat throughput and RTT tasks using pScheduler command. First the user will configure pScheduler to run a throughput task every 30 seconds. Then, the user will run an RTT task every 45 seconds. Any pScheduler task can be configured to run repeatedly by adding options to the task command:

- `--start TIMESTAMP`: it runs the first iteration of the task at `TIMESTAMP`.
- `--repeat DURATION`: Repeat runs at intervals of `DURATION`.
- `--max-runs N`: Allow the task to run up to N times.
- `--until TIMESTAMP`: Repeat runs of the task until `TIMESTAMP`.
- `--slip DURATION`: Allow the start of each run to be as much as `DURATION` later than their ideal scheduled time. If the environment variable `PSCHEDULER_SLIP` is

present, its value will be used as a default. Failing that, the default will be PT5M. Notice that the slip value also applies to non-repeating tasks.

- `--sliprand`: Randomly choose a timeslot within the allowed slip instead of choosing earliest available.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task --repeat PT20M --max-runs 10 rtt --dest 192.168.2.10
```

- `pscheduler`: is the command to interact with perfSONAR.
- `--repeat PT20M`: is a pScheduler command that configure the task to be repeated every 20 minutes.
- `task`: is a pScheduler command to specify a measurement test.
- `throughput`: is the test type.
- `--dest` is the destination node, in this case it is perfSONAR2 node (192.168.2.10). Notice that the source node is not explicit, this means that the source node is perfSONAR1 (192.168.1.10).

```
[admin@perfsonar1 ~]# pscheduler task --repeat PT20M --max-runs 10 rtt --dest 192.168.2.10
Submitting task...
Task URL:
https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-10c88a455cb6
Running with tool 'ping'
Fetching first run...

Next scheduled run:
https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-10c88a455cb6/runs/6b4ea840-a175-4e6e-8c73-316d5c1f9a56
Starts 2019-05-09T19:25:19Z (~8 seconds)
Ends 2019-05-09T19:25:38Z (~10 seconds)
Waiting for result...

 1    192.168.2.10  64 Bytes  TTL 63  RTT  0.3100 ms
 2    192.168.2.10  64 Bytes  TTL 63  RTT  0.2800 ms
 3    192.168.2.10  64 Bytes  TTL 63  RTT  0.3640 ms
 4    192.168.2.10  64 Bytes  TTL 63  RTT  0.3610 ms
 5    192.168.2.10  64 Bytes  TTL 63  RTT  0.4570 ms

0% Packet Loss  RTT Min/Mean/Max/StdDev = 0.280000/0.354000/0.457000/0.062000 ms

Next scheduled run:
https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-10c88a455cb6/runs/b14ca5b1-c399-4448-a413-b301aad5fc00
Starts 2019-05-09T19:45:19Z (~1187 seconds)
```

The figure above shows the first measurement of the round-trip time. Notice that the task is going to be repeated 10 times in 20 minutes.

Step 2. To return to the CLI, press `Ctrl+C`. Notice that the task will keep running.

4 Exporting and importing tasks to JSON

The user can export a pScheduler task to a Java Script Object Notation (JSON) file. The JSON version of a task specification can be sent to the standard output without scheduling using the `--export` command.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task --repeat PT3M --export throughput --source 192.168.1.10 --dest
192.168.2.10 > my_test_1
```

- `pscheduler`: is the command to interact with perfSONAR.
- `task`: is a pScheduler command to specify a measurement test.
- `--repeat PT3M`: is a pScheduler command that configure the task to be repeated every 3 minutes.
- `--export`: is to indicate that the task will not be executed but stored.
- `throughput`: is the test type.
- `--source`: is to specify where the test should originate, in this case it is perfSONAR1 node (*192.168.1.10*).
- `--dest` is the destination node, in this case it is perfSONAR2 node (*192.168.2.10*).
- `> my_test_1`: is to create a file where the task is going to be stored.

```
[admin@perfsonar1 ~]$ pscheduler task --repeat PT3M --export throughput --source 192.168.1.10 --dest
192.168.2.10 > my_test_1
[admin@perfsonar1 ~]$ _
```

Step 2. In order to visualize the file, type `cat my_test_1`. A JSON file will be displayed. This file contents a pScheduler task, however this task is not running. Notice also that the task might be invalid because tasks are not validated until they are submitted for scheduling.

```
[admin@perfsonar1 ~]$ cat my_test_1
{
  "schedule": {
    "repeat": "PT3M"
  },
  "schema": 1,
  "test": {
    "spec": {
      "dest": "192.168.2.10",
      "schema": 1,
      "source": "192.168.1.10"
    },
    "type": "throughput"
  }
}
[admin@perfsonar1 ~]$ _
```

Step 3. A JSON file that was previously exported or generated elsewhere can be imported using the `--import` command. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler task --import my_test_1
```

- `pscheduler`: is the command to interact with perfSONAR.
- `task`: is a pScheduler command to specify a measurement test.
- `my_test_1`: is the file that contains the task.

```

[admin@perfsonar1 ~]# pscheduler task --import mj_test_1
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/eacefe1e-b725-411a-89f4-31a4320f8712
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/eacefe1e-b725-411a-89f4-31a4320f8712/runs/1d705fe8-dc10-4be5-9af3-c585116d3010
Starts 2019-05-09T18:05:20Z (~7 seconds)
Ends 2019-05-09T18:05:39Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput    Retransmits   Current Window
0.0 - 1.0    6.83 Gbps    991           893.42 KBytes
1.0 - 2.0    6.38 Gbps    157           754.41 KBytes
2.0 - 3.0    6.80 Gbps    131           596.58 KBytes
3.0 - 4.0    6.50 Gbps    132           674.77 KBytes
4.0 - 5.0    7.06 Gbps    0             836.94 KBytes
5.0 - 6.0    6.68 Gbps    64            713.86 KBytes
6.0 - 7.0    6.62 Gbps    9             566.17 KBytes
7.0 - 8.0    4.19 Gbps    0             750.06 KBytes
8.0 - 9.0    1.90 Gbps    0             809.43 KBytes
9.0 - 10.0   6.70 Gbps    172           676.22 KBytes

Summary
Interval      Throughput    Retransmits
0.0 - 10.0    5.97 Gbps    1656

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/eacefe1e-b725-411a-89f4-31a4320f8712/runs/67a2c999-8181-40e1-884e-400bcb7e6b60
Starts 2019-05-09T18:35:20Z (~1779 seconds)

```

Step 4. To return to the CLI, press `Ctrl+C`. Notice that the task will keep running.

5 Viewing the schedule

In this section, it is presented two visualization tools, pScheduler monitor and pScheduler schedule. The tests scheduled in the last section still running. The user will use pScheduler commands to visualize the schedule.

5.1 pScheduler monitor

The `pscheduler monitor` command provides top-like output of what the schedule is doing in near real time. It takes the following form:

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler monitor
```

```
[admin@perfsonar1 ~]# pscheduler monitor
```

The user will see the scheduled tests. These tests have a status depending on whether they have already run or are still waiting to do so. Possible status values are:

- *Pending*: This run is scheduled to execute at some point in the future.

- *On Deck*: This run is scheduled to execute and will begin execution very soon.
- *Running*: This run is in the middle of execution.
- *Cleanup*: This run completed execution and is doing some final operations.
- *Finished*: The run has already executed and finished successfully.
- *Overdue*: The run was scheduled to execute at a certain time in the past but did not. It may get executed soon if it is not beyond a certain threshold.
- *Missed*: The run was scheduled but did not execute at its given time. This can happen if the scheduler was not running at the allotted time or the task was paused.
- *Failed*: The run failed to complete for some reason.
- *Non-Starter*: The run could not be scheduled because there were no timeslots that could accommodate the constraints.
- *Canceled*: The task was cancelled before the run was executed.

```

2019-05-09T14:26:24-04:00          pScheduler Monitor          perfsonar1
2019-05-09T10:23:50Z Finished      trace --dest 192.168.3.10 --source 192.168.1.10
2019-05-09T10:33:30Z Finished      trace --dest 192.168.3.10 --source 192.168.1.10
2019-05-09T11:04:18Z Finished      throughput --dest 192.168.3.10
2019-05-09T17:09:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:11:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:13:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:15:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:17:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:19:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:21:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:23:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:25:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T17:27:04Z Finished      rtt --dest 192.168.2.10
2019-05-09T18:05:39Z Finished      throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:22:32Z Finished      rtt --dest 192.168.2.10
2019-05-09T18:23:41Z Finished      throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:24:32Z Finished      rtt --dest 192.168.2.10
2019-05-09T18:26:22Z Running       throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:26:28Z On Deck      rtt --dest 192.168.2.10
2019-05-09T18:28:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:29:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:30:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:32:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:32:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:34:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:35:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:36:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:38:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:38:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:40:28Z Pending     rtt --dest 192.168.2.10
2019-05-09T18:41:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:44:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:47:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:50:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10
2019-05-09T18:53:22Z Pending     throughput --source 192.168.1.10 --dest 192.168.2.10

```

Step 2. To exit from pScheduler monitor, press `Ctrl+C`.

5.2 pScheduler schedule

The `pscheduler schedule` command asks pScheduler to fetch scheduled task runs from the past, present or future and display them as text.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler schedule
```

```
[admin@perfsonar1 ~]# pscheduler schedule
2019-05-09T18:30:28Z - 2019-05-09T18:30:39Z (On Deck)
rtt --dest 192.168.2.10 (Run with tool 'ping')
https://localhost/pscheduler/tasks/d46a6222-2ddd-48a4-906a-99df5579e212/runs/155d7165-09c0-48ad-b5da
-aaf7a991455c

2019-05-09T18:32:22Z - 2019-05-09T18:32:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/12564426-1378-4e99-b6ff
-7f650ae4d5bb

2019-05-09T18:32:28Z - 2019-05-09T18:32:39Z (Pending)
rtt --dest 192.168.2.10 (Run with tool 'ping')
https://localhost/pscheduler/tasks/d46a6222-2ddd-48a4-906a-99df5579e212/runs/e985217c-f0eb-4a1d-8a37
-be819cddcf25

2019-05-09T18:34:28Z - 2019-05-09T18:34:39Z (Pending)
rtt --dest 192.168.2.10 (Run with tool 'ping')
https://localhost/pscheduler/tasks/d46a6222-2ddd-48a4-906a-99df5579e212/runs/8327a96b-d6ca-4f2d-8fb6
-1db70b63625c

2019-05-09T18:35:22Z - 2019-05-09T18:35:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/1e932f62-e3f2-4dc8-b3bf
-4e178791c59d
```

Step 2. To exit from pScheduler schedule, press `Ctrl+C`.

6 Canceling tasks

So far there are two pscheduler tasks running. In this section, the user will cancel the scheduled Round-Trip Time (RTT) and throughput tasks which are running.

Step 1. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler schedule --filter-test rtt
```

```
[admin@perfsonar1 ~]# pscheduler schedule --filter-test rtt
2019-05-09T19:45:19Z - 2019-05-09T19:45:30Z (Pending)
rtt --dest 192.168.2.10 (Run with tool 'ping')
https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-10c88a455cb6/runs/b14ca5b1-c399-4448-a413
-b301aad5fc80

2019-05-09T20:05:19Z - 2019-05-09T20:05:30Z (Pending)
rtt --dest 192.168.2.10 (Run with tool 'ping')
https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-10c88a455cb6/runs/028d34eb-f434-4fc2-b65c
-88d44c08c349

2019-05-09T20:25:19Z - 2019-05-09T20:25:30Z (Pending)
rtt --dest 192.168.2.10 (Run with tool 'ping')
https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-10c88a455cb6/runs/de3ac9f6-4337-47b6-90c5
-dc3433b03fc2
[admin@perfsonar1 ~]#
```

The user will see the scheduled task for Round-Trip Time (RTT) measurement.

Step 2. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler cancel https://localhost/pscheduler/tasks/[url]
```

Replace `[url]` with the first three characters of the last task URL. In this example, the first three characters of the task is `631`, these characters may vary then, press *Tab* key to autocomplete the following characters. Press *Enter* to cancel the task.

```
admin@perfsonar1 ~]$ pscheduler cancel https://localhost/pscheduler/tasks/631bb0cd-f92c-4de7-be3e-1
9c88a455cb6
admin@perfsonar1 ~]$ _
```

Step 3. In perfSONAR1 command line, type the command `pscheduler monitor` to visualize the schedule.

2019-05-09T15:32:10-04:00	pScheduler Monitor		perfsonar1
2019-05-09T18:41:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:44:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:47:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:50:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:53:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:56:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:59:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:02:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:05:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:08:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:11:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:14:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:17:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:20:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:23:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:25:23Z	Finished	rtt --dest 192.168.2.10	
2019-05-09T19:26:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:29:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:32:22Z	On Deck	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:35:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:38:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:41:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:44:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:47:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:50:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:53:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:56:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:59:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:02:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:05:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:08:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:11:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:14:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:17:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T20:20:22Z	Pending	throughput --source 192.168.1.10 --dest 192.168.2.10	

The user will notice that all the Round-Trip Time (RTT) tasks are finished and there are not more tasks like this scheduled.

Step 4. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler schedule --filter-test throughput
```

The user will see the scheduled task for round-trip time (RTT) tests.

```

[admin@perfsonar1 ~]# pscheduler schedule --filter-test throughput
2019-05-09T19:35:22Z - 2019-05-09T19:35:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/8ea2bf35-207a-45d3-8843-758914649f3d

2019-05-09T19:38:22Z - 2019-05-09T19:38:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/de3d63e9-4c4a-4860-b04f-12dabdf45a6

2019-05-09T19:41:22Z - 2019-05-09T19:41:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/9fbc3b67-14a7-4ae0-a89f-a9466204db6e

2019-05-09T19:44:22Z - 2019-05-09T19:44:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/766bd658-72e2-4e22-b4c3-90bc0911c9fa

2019-05-09T19:47:22Z - 2019-05-09T19:47:41Z (Pending)
throughput --source 192.168.1.10 --dest 192.168.2.10 (Run with tool 'iperf3')
https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a4048c4b6b85/runs/1888440f-9e39-4d5c-90ef-230cd7009b4c
    
```

Step 5. In perfSONAR1 command line, follow the command format (1) and type:

```
pscheduler cancel https://localhost/pscheduler/tasks/[url]
```

Replace `[url]` with the first three characters of the last task URL. In this example, the first three characters of the task is `f44`, these characters may vary then, press *Tab* key to autocomplete the following characters. Press *Enter* to cancel the task.

```

[admin@perfsonar1 ~]# pscheduler cancel https://localhost/pscheduler/tasks/f44018ab-c6a1-4fc7-867a-a
4048c4b6b85
[admin@perfsonar1 ~]# _
    
```

Step 6. In perfSONAR1 command line, type the command `pscheduler monitor` to visualize the schedule.

2019-05-09T15:34:24-04:00	pScheduler Monitor		perfsonar1
2019-05-09T18:44:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:47:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:50:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:53:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:56:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T18:59:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:02:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:05:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:08:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:11:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:14:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:17:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:20:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:23:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:25:23Z	Finished	rtt --dest 192.168.2.10	
2019-05-09T19:26:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:29:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	
2019-05-09T19:32:41Z	Finished	throughput --source 192.168.1.10 --dest 192.168.2.10	

The user will notice that all the tasks are finished and there are not more tasks scheduled.

This concludes lab 5.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swamy and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 6: Bandwidth-delay Product and TCP Buffer Size

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	4
Lab roadmap.....	4
1 Introduction.....	4
1.1 TCP buffers.....	4
1.2 Bandwidth-delay product	5
1.3 Practical observations on setting TCP buffer size	6
1.4 TCP window size calculated value.....	6
1.5 Zero window	7
2 Emulating 2 Gbps high-latency WAN.....	7
3 BDP and buffer size experiments	11
3.1 Window size in sysctl	12
4 Modifying buffer size and throughput test	13
References.....	15

Overview

This lab explains the Bandwidth-Delay Product (BDP) in Wide Area Networks (WAN) and how to perform TCP Tuning in a perfSONAR node to modify the buffer size. Throughput measurements are also conducted in this lab to verify the buffer size configuration using pScheduler commands.

Objectives

By the end of this lab, the user will:

1. Understand Bandwidth-Delay Product (BDP).
2. Define TCP window size.
3. TCP window size calculation.
4. Change buffer size with *sysctl*.
5. Emulate WAN using NETEM commands.
6. Visualize the results on pScheduler report.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

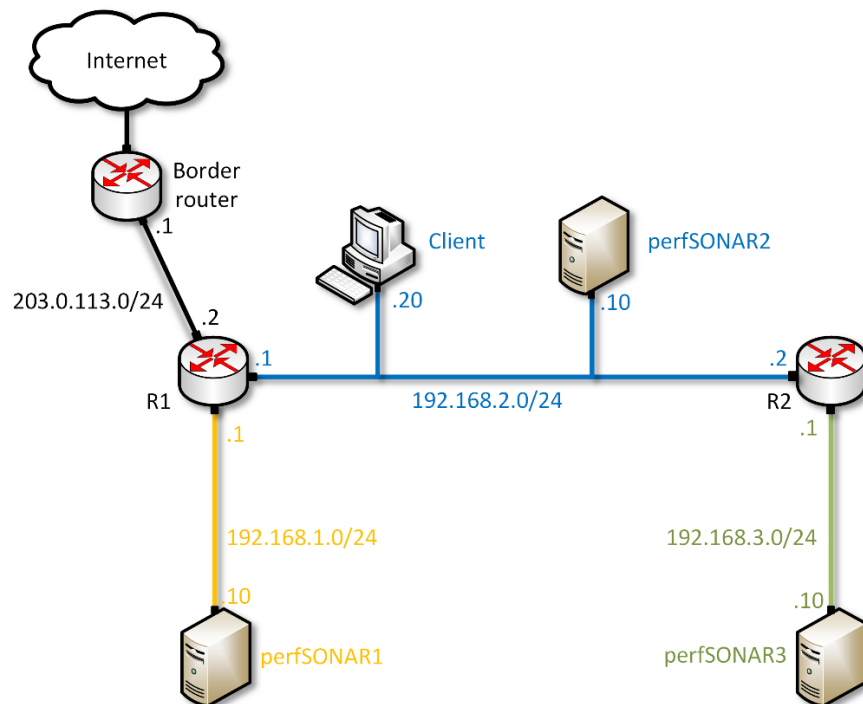


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

The lab includes the following tasks:

1. Section 1: Introduction.
2. Section 2: Emulating 2 Gbps high-latency WAN.
3. Section 3: BDP and buffer size experiments.
4. Section 4: Modifying the buffer size and throughput test.

1 Introduction

1.1 TCP buffers

Consider Figure 1, which shows a data transfer between a sender and a receiver. At the sender side, TCP receives data from the application layer and places it in the TCP buffer. Typically, TCP fragments the data in the buffer into Maximum Segment Size (MSS) units and passes the newly formed segments (application-layer data plus TCP header) to the network layer. In this example, the MSS is 100 bytes. A segment stored in the TCP send buffer will only be removed from the buffer when a corresponding acknowledgement is received. If the send buffer is full, TCP blocks the application from sending new data. Each segment carries a sequence number, which is the byte-stream number of the first byte in the segment. The corresponding acknowledgement (Ack) carries the number of the next expected byte (e.g., Ack-101 acknowledges bytes 1-100, carried by the first segment).

At the receiver side, TCP receives data from the network layer and places it into the TCP receive buffer. TCP delivers the data *in order* to the application layer. The implication here is that bytes contained in a segment, say segment 2 (bytes 101-200), cannot be delivered to the application layer before the bytes contained in segment 1 (bytes 1-100) are

delivered to the application layer. At any given time, the TCP receiver indicates to the TCP sender how many bytes the latter can send. This reflects how much free buffer space is available at the receiver.

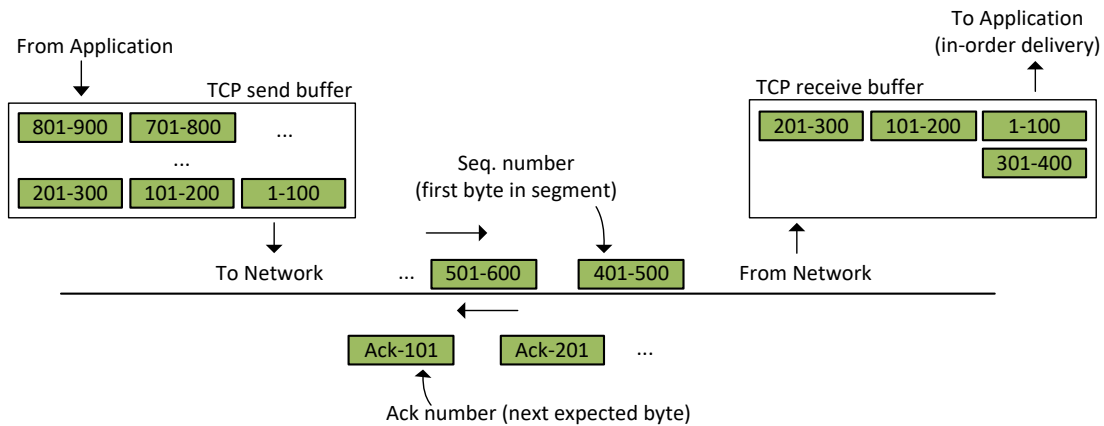


Figure 1. TCP send and receive buffers.

1.2 Bandwidth-delay product

In many Wide Area Networks (WANs) connecting geographically separated locations, the Round-Trip Time (RTT), which is the time it takes for a small packet to travel from sender to receiver and then back to the sender, is dominated by the propagation delay. Long RTTs along with TCP buffer size can have important implications for the efficiency of the bandwidth utilization and throughput. As an example, consider a 10 Gbps WAN with a 50-millisecond RTT. Assume that the TCP send and receive buffer sizes are set to 1 Mbyte (1 Mbyte = 1024^2 bytes = 1,048,576 bytes = $1,048,576 \cdot 8$ bits = 8,388,608 bits). At 10 Gbps, this number of bits is approximately transmitted in:

$$T_{tx} = \frac{\text{\# bits}}{\text{transmission rate}} = \frac{8,388,608}{10 \cdot 10^9} = 0.84 \text{ milliseconds.}$$

I.e., if at $t = 0$ the TCP sender starts transmitting, at $t = 0.84$ milliseconds the content in TCP send buffer has been completely sent. At this point, TCP must wait for the corresponding acknowledgements, which will only start arriving at $t = 50$ milliseconds. This means that the sender only uses $0.84/50$ or 1.68% of the available bandwidth.

The solution to that above problem lies in allowing the sender to continuously transmit segments until the corresponding acknowledgments arrive back. Note that the first acknowledgement arrives after $RTT = 50$ milliseconds. The number of bits that can be transmitted in this time period is given by bandwidth of the channel in bits per second multiplied by the RTT. This quantity is referred to as the Bandwidth-Delay Product (BDP). For the above example, the buffer size must be greater than or equal to the BDP:

$$\text{TCP buffer size} \geq \text{BDP} = (10 \cdot 10^9)(50 \cdot 10^{-3}) = 500,000,000 \text{ bits} = 62,500,000 \text{ bytes.}$$

The first factor ($10 \cdot 10^9$) is the bandwidth; the second factor ($50 \cdot 10^{-3}$) is the RTT. For practical purposes/configuration, the TCP buffer can be also expressed in Mbytes (1

Mbyte = 1024^2 bytes) or Gbytes (1 Gbyte = 1024^3 bytes). The above expression, in Mbytes, is:

$$\text{TCP buffer size} \geq 62,500,000 \text{ bytes} = 59.6 \text{ Mbytes} \approx 60 \text{ Mbytes.}$$

1.3 Practical observations on setting TCP buffer size

Linux systems configuration. When configuring the buffer size in Linux systems, it is important to note that Linux assumes that half of the send/receive TCP buffers are used for internal kernel structures. Thus, only half of the buffer size is used to store segments. This implies that if a TCP connection requires certain buffer size, then the administrator must configure the buffer size equals to twice the bandwidth-delay product. For the previous example, the TCP buffer size must be:

$$\text{TCP buffer size} \geq 2 \cdot \text{BDP} = 2 \cdot 60 \text{ Mbytes} = 120 \text{ Mbytes.}$$

Packet loss scenarios. TCP provides a reliable, in-order delivery service. In this context, reliability means that bytes successfully received must be acknowledged. The sender will only release (free the memory) a segment stored in its TCP send buffer after it receives the corresponding acknowledgement. In-order delivery means that the receiver only delivers bytes to the application layer in sequential order. This has some performance implications, as illustrated next. Consider Figure 2, which shows a TCP receive buffer. Assume that the segment carrying bytes 101-200 was lost in transit. Although the receiver has successfully received bytes 301-900, they cannot be delivered to the application layer until bytes 101-200 are received. Note that the receive buffer may become full, thus preventing the reception of additional bytes beyond byte 900. Thus, the sender will be blocked, and the bandwidth will be underutilized (eventually, the sender will retransmit the segment 101-200).

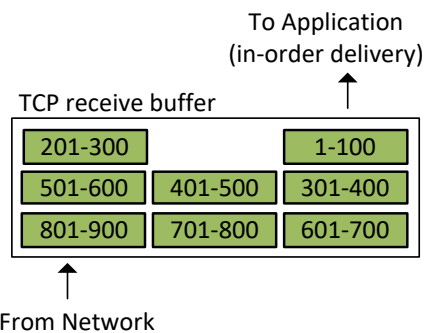


Figure 2. TCP receive buffer. Although bytes 301-900, they cannot be delivered to the application layer until the segment carrying bytes 101-200 are received.

To fully utilize the available bandwidth, the TCP send and receive buffer must be large enough to prevent such situation.

1.4 TCP window size calculated value

The receiver must constantly communicate with the sender to indicate how much free buffer space is available in the TCP receive buffer. This information is carried in a TCP header field called window size. The window size has a maximum value of 65,535 bytes, as the header value allocated for the window size is two bytes long (16 bits; $2^{16}-1 = 65,535$). However, this value is not large enough for high-bandwidth high-latency networks. Therefore, *TCP window scale option* was standardized in RFC 1323. By using this option, the calculated window size may be increased up to a maximum value of 1,073,725,440 bytes.

When advertising its window, a device also advertises the *scale factor* (multiplier) that will be used throughout the session. The TCP window size is calculated as follows:

$$\text{Scaled TCP}_{\text{Win}} = \text{TCP}_{\text{Win}} \cdot \text{Scaling Factor}$$

Consider the following example. For an advertised TCP window of 2,049 and a scale factor of 512, then the final window size is 1,049,088 bytes. Figure 3 displays a packet inspected in Wireshark protocol analyzer for this numerical example.

```

▶ Flags: 0x010 (ACK)
  Window size value: 2049
  [Calculated window size: 1049088]
  [Window size scaling factor: 512]

```

Figure 3. Window Scaling in Wireshark.

1.5 Zero window

When the TCP buffer is full, a window size of zero is advertised to inform the other end that it cannot receive any more data. When a client sends a TCP Window of zero, the server will pause its data transmission, and waits for the client to recover. Once the client is recovered, it digests data and informs the server to resume the transmission by setting again the TCP Window.

2 Emulating 2 Gbps high-latency WAN

In this section, the user will emulate a high-latency WAN by introducing a 100ms delay to the network. Specifically, the user will set 50ms delay to the router R1 and 50ms delay to router R2 using Network Emulator (NETEM) commands. Additionally, the bandwidth between perfSONAR1 and perfSONAR3 nodes will be set to 2 Gbps using Token Bucket Filter (TBF). In order to verify, the user will run a throughput test using pScheduler commands.

Step 1. On the topology, click on router R1 and enter the username `root` and `password` as password. Note that the password will not be displayed while typing it.


```
CentOS Linux 7 (Core)
Kernel 4.19.1-1.el7.elrepo.x86_64 on an x86_64

R1 login: root
Password:
```

Step 2. To identify the interface connected to the network *192.168.2.0/24*, in router R1 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R1 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::1db9:17ab:218f:23a7 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:9a:bd txqueuelen 1000 (Ethernet)
    RX packets 38 bytes 2753 (2.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 32 bytes 2839 (2.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens36: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 203.0.113.2 netmask 255.255.255.0 broadcast 203.0.113.255
    inet6 fe80::c1a0:d0ce:dde9:24ce prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:8f:fa txqueuelen 1000 (Ethernet)
    RX packets 128 bytes 12751 (12.4 KiB)
    RX errors 0 dropped 4 overruns 0 frame 0
    TX packets 235 bytes 18898 (18.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.1 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::46d7:42e:b419:21a1 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e4:84 txqueuelen 1000 (Ethernet)
    RX packets 130 bytes 10161 (9.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 73 bytes 6898 (6.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R1 ~]# _
```

The output of the `ifconfig` indicates that router R1 has three interfaces. The interface *ens37* connects router R1 to the network *192.168.2.0/24*. Thus, this interface must be used for emulation.

Step 3. In order to add a 50ms delay, in router R1 CLI type the following command:

```
sudo tc qdisc add dev ens37 root handle 1: netem delay 50ms
```

```
[root@R1 ~]# sudo tc qdisc add dev ens37 root handle 1: netem delay 50ms
[root@R1 ~]#
```

Step 4. In order to set the bandwidth, type the command shown below. This command sets the bandwidth to 2 Gbps on router R1 *ens37* interface. The `tbfb` parameters are the following:

- `rate`: 2gbit
- `burst`: 500,000
- `limit`: 50,000,000

```
sudo tc qdisc add dev ens37 parent 1: handle 2: tbf rate 2gbit burst 500000
limit 50000000
```

```
[root@R1 ~]# sudo tc qdisc add dev ens37 parent 1: handle 2: tbf rate 2gbit burst 500000 limit 50000000
[root@R1 ~]#
```

Step 5. On the topology, click on R2 and enter the username `root` and `password` as password. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 4.19.1-1.el7.elrepo.x86_64 on an x86_64

R2 login: root
Password:
```

Step 6. To identify the interface connected to the network `192.168.2.0/24`, in R2 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R2 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.1 netmask 255.255.255.0 broadcast 192.168.3.255
    inet6 fe80::250:56ff:feae:e5dc prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e5:dc txqueuelen 1000 (Ethernet)
    RX packets 1013392857 bytes 1480074592733 (1.3 TiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1217711915 bytes 2909379957450 (2.6 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.2 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::10a6:2962:4b7e:c21a prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:96:6a txqueuelen 1000 (Ethernet)
    RX packets 1213137503 bytes 1799533693195 (1.6 TiB)
    RX errors 0 dropped 10 overruns 0 frame 0
    TX packets 1016660699 bytes 2402663209617 (2.1 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 468 bytes 37528 (36.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 468 bytes 37528 (36.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R2 ~]#
```

The output of the `ifconfig` command indicates that R2 has two interfaces. The interface `ens37` connects R2 to the network `192.168.2.0/24`. Thus, this interface must be used for emulation.

Step 7. In order to add a 50ms delay, in R2 command line type the command:

```
sudo tc qdisc add dev ens37 root netem delay 50ms
```

```
[root@R2 ~]# sudo tc qdisc add dev ens37 root netem delay 50ms
[root@R2 ~]#
```

Step 8. To verify if the parameters were applied, on the topology, click on perfSONAR1 node and login entering the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar1 login: admin
Password:
Last login: Wed Jan 30 15:14:47 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/mptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar1 ~]#
```

Step 9. In perfSONAR1 command line, type the following command to verify if the delay was applied:

```
pscheduler task rtt --source 192.168.1.10 --dest 192.168.3.10
```

- `pscheduler` is the command to interact with perfSONAR.
- `task` is a pScheduler command.
- `rtt` is the test type.
- `--source` is to specify where the test should originate, in this case it is perfSONAR1 node (`192.168.1.10`).
- `--dest` is the destination node, in this case it is perfSONAR3 (`192.168.3.10`).

```
[admin@perfsonar1 ~]# pscheduler task rtt --source 192.168.1.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/68fa346d-b028-4419-9804-1d189b5ae252
Running with tool 'ping'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/68fa346d-b028-4419-9804-1d189b5ae252/runs/7c8b57dd-5faf-4074-9265-5e34be967b76
Starts 2020-01-03T21:51:38Z (~7 seconds)
Ends 2020-01-03T21:51:49Z (~10 seconds)
Waiting for result...

1 192.168.3.10 64 Bytes TTL 62 RTT 100.0000 ms
2 192.168.3.10 64 Bytes TTL 62 RTT 100.0000 ms
3 192.168.3.10 64 Bytes TTL 62 RTT 100.0000 ms
4 192.168.3.10 64 Bytes TTL 62 RTT 100.0000 ms
5 192.168.3.10 64 Bytes TTL 62 RTT 100.0000 ms

0% Packet Loss RTT Min/Mean/Max/StdDev = 100.530000/100.598000/100.761000/0.217000 ms

No further runs scheduled.
[admin@perfsonar1 ~]#
```

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 100.53, 100.59, 100.761 and 0.217 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 100ms.

Step 10. To verify the throughput, in perfSONAR1 command line, type the following command:

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.3.10
```

- `pscheduler` is the command to interact with perfSONAR.
- `task` is a pScheduler command.
- `throughput` is the test type.
- `--source` is to specify where the test should originate, in this case it is perfSONAR1 node (192.168.1.10).
- `--dest` is the destination node, in this case it is perfSONAR3 (192.168.3.10).

```
[admin@perfsonar1 ~]# pscheduler task throughput --source 192.168.1.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/4c56e9ff-9f6d-428f-8d53-e42a20a122ff
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/4c56e9ff-9f6d-428f-8d53-e42a20a122ff/runs/d564a625-b91b-4e75-b865-bde6e62c3ef0
Starts 2020-01-03T22:03:41Z (~6 seconds)
Ends 2020-01-03T22:04:00Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput      Retransmits      Current Window
0.0 - 1.0     144.49 Mbps     0                 3.50 MBytes
1.0 - 2.0     1.06 Gbps       1945              17.70 MBytes
2.0 - 3.0     1.42 Gbps       0                 17.70 MBytes
3.0 - 4.0     1.41 Gbps       0                 17.75 MBytes
4.0 - 5.0     1.39 Gbps       0                 17.86 MBytes
5.0 - 6.0     1.14 Gbps       222               8.98 MBytes
6.0 - 7.0     702.55 Mbps     0                 9.00 MBytes
7.0 - 8.0     702.55 Mbps     0                 9.12 MBytes
8.0 - 9.0     734.00 Mbps     0                 9.38 MBytes
9.0 - 10.0    765.44 Mbps     0                 9.75 MBytes

Summary
Interval      Throughput      Retransmits
0.0 - 10.0    946.62 Mbps     2167

No further runs scheduled.
[admin@perfsonar1 ~]#
```

Notice the measured throughput now is around 950 Mbps, which is different than the value assigned in the `tbf` rule. In the following section, the user will modify the send and receive TCP buffer size in order to achieve 2 Gbps bandwidth.

3 BDP and buffer size experiments

In a connection-oriented protocol such as TCP, BDP plays an important role as it represents the amount of buffering required on both senders and receivers (transmitting

end-hosts). In connections that have a small BDP (either because the link has a low bandwidth or because the latency is small), buffers are usually small. However, in high-bandwidth high-latency networks, where the BDP is large, a larger buffer is required to achieve the maximum theoretical bandwidth.

3.1 Window size in sysctl

In this section, the user will set different values the corresponding *sysctl* keys, which is used for dynamically changing parameters in the Linux operating system. It allows users to modify kernel parameters dynamically without rebuilding the Linux kernel.

The *sysctl* key for the receive window size is `net.ipv4.tcp_rmem` and the send window size is `net.ipv4.tcp_wmem`.

Step 1. To read the current receiver window size value of perfSONAR1 node, type the following command:

```
sysctl net.ipv4.tcp_rmem
```

```
[admin@perfsonar1 ~]# sysctl net.ipv4.tcp_rmem
net.ipv4.tcp_rmem = 4096      87380    33554432
[admin@perfsonar1 ~]# _
```

Step 2. To read the current send window size value of perfSONAR1 node, type the following command:

```
sysctl net.ipv4.tcp_wmem
```

```
[admin@perfsonar1 ~]# sysctl net.ipv4.tcp_wmem
net.ipv4.tcp_wmem = 4096      65536    33554432
[admin@perfsonar1 ~]#
```

The returned values of both keys (`net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem`) are measured in bytes. The first number represents the minimum buffer size used by each TCP socket. The middle one is the default buffer which is allocated when applications create a TCP socket. The last one is the maximum receive buffer that can be allocated for a TCP socket. Note that similar results are displayed in perfSONAR2 and perfSONAR3. For simplicity, in this section is just shown the values on perfSONAR1 node.

The default values used by in perfSONAR nodes are:

- Minimum: 4,096
- Default: 65,536
- Maximum: 33,554,432

Note that the maximum value is 32 Mbytes. However, to achieve the maximum throughput, it is necessary to set the send and receive TCP buffer size to at least twice

Bandwidth-Delay Product ($2 \cdot BDP$). In the previous test (2 Gbps, 100ms delay), the buffer size was not modified on end-hosts namely, perfSONAR1 and perfSONAR3 nodes.

The BDP for the above test is:

$$BDP = (2 \cdot 10^9) \cdot (100 \cdot 10^{-3}) = 200,000,000 \text{ bits} = 25,000,000 \text{ bytes} \approx 25 \text{ Mbytes.}$$

Note that twice BDP is around 50 Mbytes thus, this value is significantly greater than the maximum buffer size (32 Mbytes), and therefore, the pipe is not getting filled, which leads to network resources underutilization (see section 1.3).

4 Modifying buffer size and throughput test

This section repeats the throughput test after modifying the buffer size on perfSONAR1 and perfSONAR3 nodes according to the formula described above. This test assumes the same network parameters introduced in the previous test therefore, the bandwidth is limited to 2 Gbps and the RTT (delay or latency) is 100ms. The send and receive buffer sizes should be set to at least $2 \cdot BDP$. Use 25 Mbytes value for the BDP instead of 25,000,000 bytes (1 Mbyte = 1024^2 bytes).

$$BDP = 25 \text{ Mbytes} = 25 \cdot 1024^2 \text{ bytes} = 26,214,400 \text{ bytes}$$

$$\text{TCP buffer size} = 2 \cdot BDP = 2 \cdot 26,214,400 \text{ bytes} = 52,428,800 \text{ bytes}$$

Step 1. To change the TCP receive-window size value, type the following command on perfSONAR1 CLI. If a password is required, type `admin` as the password. Note that the password will not be displayed while typing it. The values set are: 4,096 (minimum), 65,536 (default) and 52,428,800 (maximum, calculated by doubling the BDP).

```
sudo sysctl -w net.ipv4.tcp_rmem='4096 65536 52428800'
```

```
admin@perfsonar1 ~1$ sudo sysctl -w net.ipv4.tcp_rmem='4096 65536 52428800'
net.ipv4.tcp_rmem = 4096 65536 52428800
admin@perfsonar1 ~1$
```

The returned values are measured in bytes. 4,096 represents the minimum buffer size that is used by each TCP socket. 65,536 is the default buffer which is allocated when applications create a TCP socket. 52,428,800 is the maximum receive buffer that can be allocated for a TCP socket.

Step 2. To change the current send-window size value, type the following command on perfSONAR1 CLI. The values set are: 4,096 (minimum), 65,536 (default) and 52,428,800 (maximum, calculated by doubling the BDP).

```
sudo sysctl -w net.ipv4.tcp_wmem='4096 65536 52428800'
```

```
admin@perfsonar1 ~1$ sudo sysctl -w net.ipv4.tcp_wmem='4096 65536 52428800'
net.ipv4.tcp_wmem = 4096 65536 52428800
admin@perfsonar1 ~1$ _
```

Step 3. To verify if the parameters were applied, on the topology, click on perfSONAR3 node and login entering the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar3 login: admin
Password:
Last login: Wed Jan 30 16:55:14 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/mptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar3 ~]$_
```

Step 4. To change the TCP receive-window size value, type the following command on perfSONAR1 CLI, if a password is required, type `admin` as password. Note that the password will not be displayed while typing it. The values set are: 4,096 (minimum), 65,536 (default) and 52,428,800 (maximum, calculated by doubling the BDP).

```
sudo sysctl -w net.ipv4.tcp_rmem='4096 65536 52428800'
```

```
[admin@perfsonar3 ~]$_ sudo sysctl -w net.ipv4.tcp_rmem='4096 65536 52428800'
net.ipv4.tcp_rmem = 4096 65536 52428800
[admin@perfsonar3 ~]$_
```

The returned values are measured in bytes. 4,096 represents the minimum buffer size that is used by each TCP socket. 65,536 is the default buffer which is allocated when applications create a TCP socket. 52,428,800 is the maximum receive buffer that can be allocated for a TCP socket.

Step 5. To change the current send-window size value, type the following command on perfSONAR1 CLI. The values set are: 4,096 (minimum), 65,536 (default) and 52,428,800 (maximum, calculated by doubling the BDP).

```
sudo sysctl -w net.ipv4.tcp_wmem='4096 65536 52428800'
```

```
[admin@perfsonar3 ~]$_ sudo sysctl -w net.ipv4.tcp_wmem='4096 65536 52428800'
net.ipv4.tcp_wmem = 4096 65536 52428800
[admin@perfsonar3 ~]$_
```

Step 6. To verify if after the configuration the throughput is achieved, go back to perfSONAR1 CLI and type the following command:

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.3.10
```

- `pscheduler` is the command to interact with perfSONAR.
- `task` is a pScheduler command.

- `throughput` is the test type.
- `--source` is to specify where the test should originate, in this case it is perfSONAR1 node (192.168.1.10).
- `--dest` is the destination node, in this case it is perfSONAR3 (192.168.3.10).

```

admin@perfsonar1 ~1$ pscheduler task throughput --source 192.168.1.10 --dest 192.168.3.10
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/8183667f-fdba-4ec9-83cb-e41e3e7ac6a7
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/8183667f-fdba-4ec9-83cb-e41e3e7ac6a7/runs/e4bd4d43-fe470e-82b7e1871be9
Starts 2020-01-03T22:16:50Z (~6 seconds)
Ends 2020-01-03T22:17:09Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput      Retransmits      Current Window
0.0 - 1.0    149.84 Mbps     0                 3.58 MBytes
1.0 - 2.0    1.69 Gbps       0                 52.48 MBytes
2.0 - 3.0    1.88 Gbps       0                 52.48 MBytes
3.0 - 4.0    1.92 Gbps       0                 52.48 MBytes
4.0 - 5.0    1.01 Gbps       196               26.24 MBytes
5.0 - 6.0    1.73 Gbps       0                 26.24 MBytes
6.0 - 7.0    1.91 Gbps       0                 26.24 MBytes
7.0 - 8.0    1.92 Gbps       0                 26.24 MBytes
8.0 - 9.0    1.91 Gbps       0                 26.24 MBytes
9.0 - 10.0   1.91 Gbps       0                 26.24 MBytes

Summary
Interval      Throughput      Retransmits
0.0 - 10.0    1.60 Gbps       196

No further runs scheduled.
admin@perfsonar1 ~1$ _

```

Note that the measured throughput now is approximately 2 Gbps, which is similar to the value assigned in the `tbw` rule (2 Gbps).

This concludes Lab 6.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swany and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.

6. M. Feit, "CLI user's guide," [Online]. Available:
<https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available:
<https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 7: Configuring Regular Tests Using a pSConfig Template

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	3
Lab roadmap.....	4
1 Introduction.....	4
2 The pSConfig file structure	5
2.1 Addresses.....	6
2.2 Groups.....	7
2.2.1 Mesh	7
2.2.2 Disjoint	8
2.2.3 List.....	9
2.3 Tests.....	10
2.4 Schedule.....	10
2.5 Archives.....	11
2.6 Tasks	11
3 Publishing a pSConfig template	13
4 Running the pSConfig pScheduler Agent.....	14
5 Viewing Scheduled Tasks	15
References.....	17

Overview

This lab presents a template framework for describing and configuring pScheduler tasks known as pSConfig. The lab describes the steps to create and interpret the parts of a pSConfig template. The user will use this template to run regular test in a perfSONAR node.

Objectives

By the end of this lab, the user will:

1. Understand the structure of pSConfig template.
2. Create a pSConfig configuration file.
3. Publish a pSConfig configuration file.
4. Verify tasks using pScheduler monitor.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

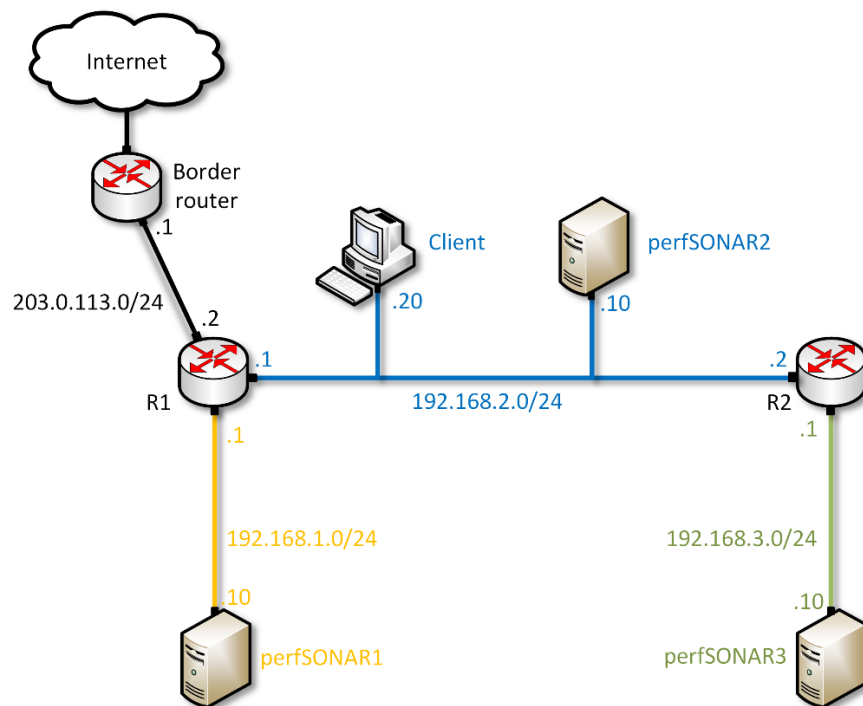


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

The lab includes the following tasks:

1. Section 1: Introduction.
2. Section 2: The pSConfig file structure.
3. Section 3: Publishing a pSConfig template.
4. Section 4: Running the pSConfig pScheduler Agent.
5. Section 5: Viewing Scheduled Tasks.

1 Introduction

The perfSONAR Configuration (pSConfig) is a template-based setup for describing and configuring a topology of tasks. A task is defined as a measurement test to be performed using pScheduler. The topology defines how multiple tasks are interrelated. Overall, the goal of pSConfig is to simplify the scheduling of such tasks as well as maintenance of visualization components when managing more than one perfSONAR node.

The basic pSConfig workflow is shown in Figure 2 and consists of three key steps:

1. A template file is defined using the machine-readable JavaScript Object Notation (JSON) file format to describe the task topology of the perfSONAR hosts.
2. This file is then published to the web where it is read by an agent to perform specific operations.
3. Agents read the pSConfig template. An agent is a software that reads one or more pSConfig templates and uses the information to perform a specific function. The pScheduler agent which reads template files and submits related measurement tests to pScheduler and, the MaDDash agent, which reads template files to generate dashboard in order display the measurement results. Overall this lab focuses on the first step of building a sample JSON template file.

In this lab, the user will understand how the file is created. Secondly, the user will publish the template in order to be accessible by any node in the topology. Then, the template will be read by an agent. Finally, the user will verify the tasks using pScheduler monitor.

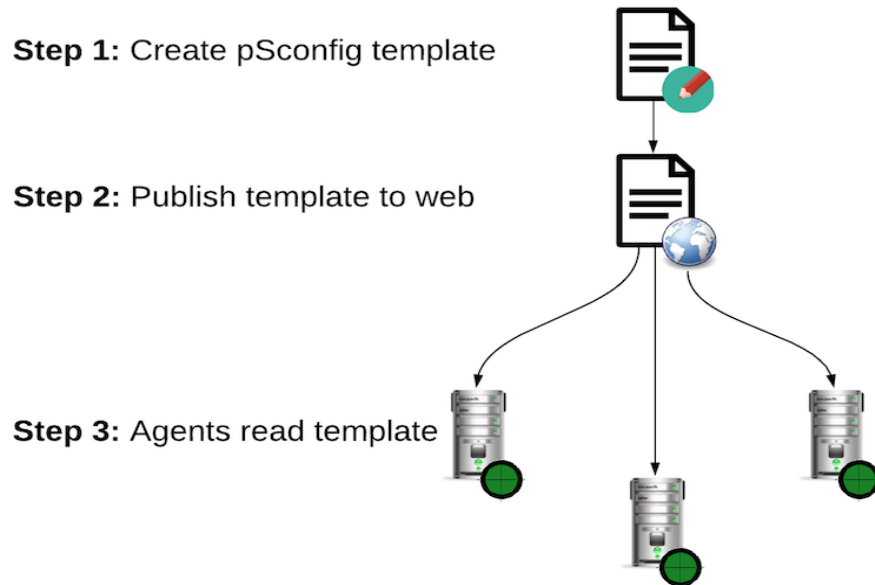


Figure 2. pSConfig workflow¹.

2 The pSConfig file structure

Basically, a pSConfig file has six components: addresses, groups, tests, archives, schedules and tasks. These components are put all together to perform measurement tests. In this section all these components are explained. For that purpose, there is available a pSConfig template in perfSONAR2 node. The user will not modify this file during the lab. This file will be useful to explain the structure of pSConfig template. In addition, this template will be used by an agent to run measurement test in the current lab topology.

Step 1. In order to visualize the pSConfig template, open perfSONAR2 and enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar2 login: admin
Password:
Last login: Wed Jun 12 22:40:24 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar2 ~]$ _
```

Step 2. After login, type the command shown below, a pSConfig template will be displayed. Use the arrows to scroll up/down into the file.

```
nano /home/template.json
```

```
GNU nano 2.3.1      File: /home/template.json

{
  "_meta": {
    "display-name": "perfSONAR Lab"
  },
  "archives": {
    "esmond_archive_1": {
      "archiver": "esmond",
      "data": {
        "measurement-agent": "% scheduled_by_address %",
        "url": "https://192.168.2.10/esmond/perfsonar/archive/"
      }
    }
  },
  "addresses": {
    "perfSONAR1": { "address": "192.168.1.10" },
    "perfSONAR2": { "address": "192.168.2.10" },
    "perfSONAR3": { "address": "192.168.3.10" }
  },
  "groups": {
    "loss_group": {
      "type": "mesh",
      "addresses": [
        { "name": "perfSONAR1" },
        { "name": "perfSONAR2" },
        { "name": "perfSONAR3" }
      ]
    },
    "throughput_group": {
      "type": "mesh",

```

Read 89 lines (Converted from DOS format - Warning: No write permission)

^G Get Help	^O WriteOut	^R Read File	^Y Prev Page	^K Cut Text	^C Cur Pos
^X Exit	^J Justify	^W Where Is	^U Next Page	^U UnCut Text	^T To Spell

Step 3. Press `Ctrl+X` to close the window. The user could repeat the previous steps anytime along this lab.

2.1 Addresses

The address is the most basic unit of a template. An address is a collection of properties that act as the unit of input to a task. This address is not necessarily connected to an interface or host. It is simply an object with properties. The figure 3 illustrates the idea.

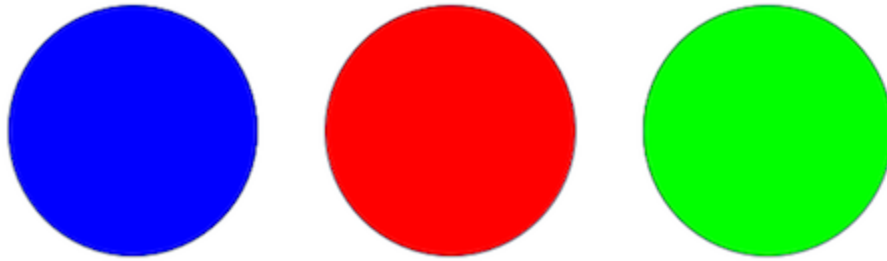


Figure 3: Representation of three addresses².

Each shape has certain properties such as the form and the color. In pSConfig these properties can be used when constructing a task.

Step 1. In perfSONAR2 type `nano /home/template.json`, a pSConfig template will be displayed.

Step 2. Go to the line 17. To see the actual line number press `Ctrl+C`.

```
"addresses": {
  "perfSONAR1": { "address": "192.168.1.10" },
  "perfSONAR2": { "address": "192.168.2.10" },
  "perfSONAR3": { "address": "192.168.3.10" }
},
```

In this configuration file these addresses are specified with the names and IP addresses of perfSONAR1, perfSONAR2 and perfSONAR3.

2.2 Groups

A group describes the way to combine addresses when building the list of tasks. All groups have a type that provides the base for how addresses are combined. Currently, pSConfig support three group types.

2.2.1 Mesh

The first group type is *mesh*. A group of type mesh pairs every address against every other address in the group. A mesh group is shown in the figure 4. Notice that the three addresses are paired in six groups.

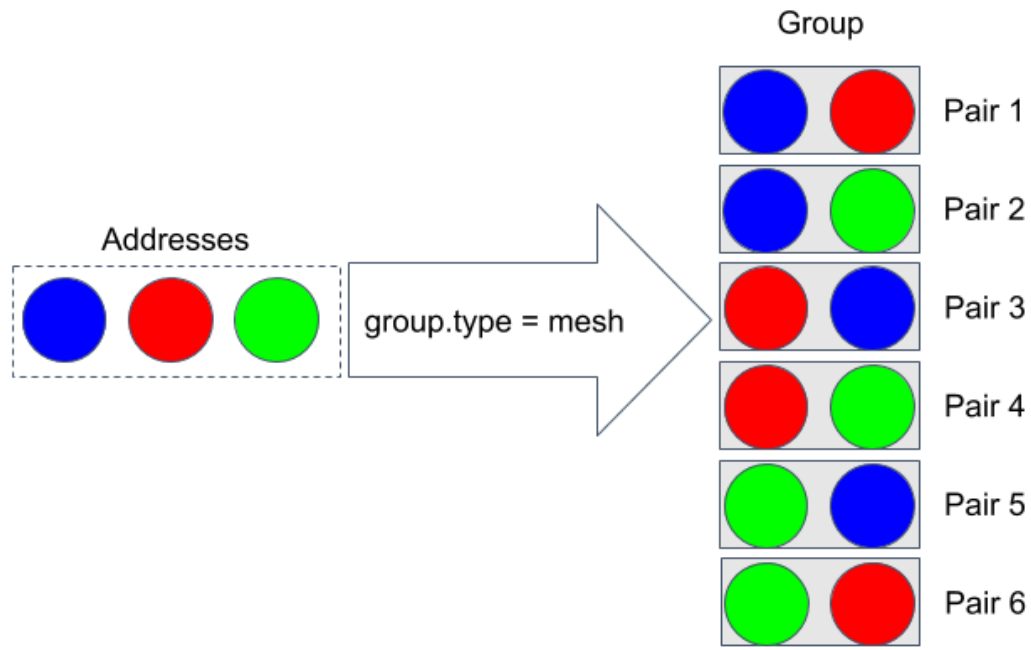


Figure 4: Mesh group².

The user could visualize the group configuration following the next steps:

Step 1. In perfSONAR2 type `nano /home/template.json`, a pSConfig template will be displayed.

Step 2. Go to the line 22. To see the actual line number press `Ctrl+C`.

```

"groups": {
  "loss_group": {
    "type": "mesh",
    "addresses": [
      { "name": "perfSONAR1" },
      { "name": "perfSONAR2" },
      { "name": "perfSONAR3" }
    ]
  },
  "throughput_group": {
    "type": "mesh",
    "addresses": [
      { "name": "perfSONAR1" },
      { "name": "perfSONAR2" },
      { "name": "perfSONAR3" }
    ]
  }
},

```

The user will see there are two groups, *loss_group* and *throughput_group*. The group type for both is *mesh* and the nodes involved in these groups are perfSONAR1, perfSONAR2 and perfSONAR3.

2.2.2 Disjoint

The second group type is *disjoint*. A group of type disjoint pairs every address in one group (Group A) with every address in another group (Group B). Both groups can have one or

more addresses. A mesh group is shown in the figure 5. Notice that the three addresses are paired in four groups.

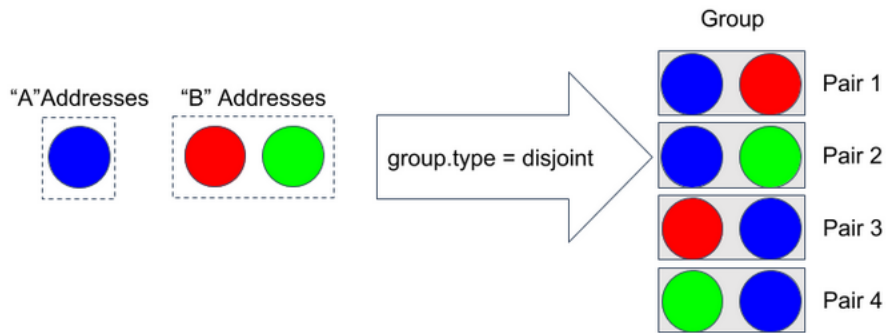


Figure 5: Disjoint group².

The pSconfig template of this lab does not provide an example for this type of group. However, a JSON object for this type of group is described below.

```
{
  "type": "disjoint",
  "a-addresses": [
    {"name": "circle1"}
  ],
  "b-addresses": [
    {"name": "circle2"},
    {"name": "circle3"}
  ]
}
```

2.2.3 List

The final group type is called *list*. A group of type list returns each address independently. It is the only current type that does not pair addresses. Instead, it just generates a one-dimensional list of addresses. The figure 5 illustrate a list group type.

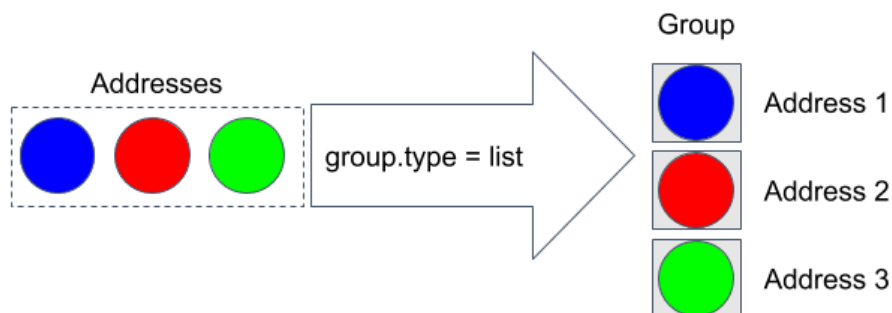


Figure 6: List group².

The pSconfig template of this lab does not provide an example for this type of group. However, a JSON object for this type of group is described below.

```
{
  "type": "list",
```

```

    "addresses": [
      {"name": "perfSONAR1"},
      {"name": "perfSONAR2"},
      {"name": "perfSONAR3"}
    ]
  }

```

2.3 Tests

Test objects define the parameters of the job to be carried out by the task. These parameters are interpreted by the agent and then it is delivered to pScheduler to run the task.

Step 1. In perfSONAR2 type `nano /home/template.json`, a pSConfig template will be displayed.

Step 2. Go to the line 41. To see the actual line number press `Ctrl+C`.

```

"tests": {
  "throughput_test": {
    "type": "throughput",
    "spec": {
      "source": "{% address[0] %}",
      "dest": "{% address[1] %}",
      "duration": "PT10S"
    }
  },
  "loss_test": {
    "type": "rtt",
    "spec": {
      "source": "{% address[0] %}",
      "dest": "{% address[1] %}"
    }
  }
}

```

The user will see there are two tests, *throughput_test* and *loss_test*. The specification for the *throughput_test* is given by the source and destination addresses. The value of these keys are *address[0]* and *address[1]* respectively. These values represent the addresses in the group pairs. Finally, the duration of the test is specified with the key *duration*, in this case the value is *PT10S* which means the test duration will last 10 seconds. The *loss_test* uses the toll *rtt* to measure the packet loss ratio. The specification indicates that the source and destination addressed will be taken also from each group pair.

2.4 Schedule

Schedule objects tell the agent how often a task will be scheduled and how long the test is going to inactive after each run. Schedule objects are borrowed directly from pScheduler.

Step 1. In perfSONAR2 type `nano /home/template.json`, a pSConfig template will be displayed.

Step 2. Go to the line 61. To see the actual line number press `Ctrl+C`.

```
"schedules": {
  "schedule_PT2M": {
    "repeat": "PT2M",
    "sliprand": true,
    "slip": "PT2M"
  }
},
```

In the pSConfig template is defined one schedule named *schedule_PT2M* which will tell a task using it to run on a random interval between every 2-4 minutes. The *repeat* property is an ISO8601 duration telling a task that uses it to repeat at least every two minutes. The *slip* says that it can run up to 2 minutes later than that (i.e. 4 minutes). Finally, *sliprand* tells it to randomly choose an interval between those two values for each run. This is commonly done to prevent tests from bunching together at the beginning of a time interval.

2.5 Archives

Archive objects are optional components of the template that tell agents where the results of the described tasks are to be stored. Archive objects at a minimum have an archiver field that indicates the type of archive and a data field containing archive-specific parameters. Archive objects in pSConfig are taken directly from pScheduler.

Step 1. In perfSONAR2 type `nano /home/template.json`, a pSConfig template will be displayed.

Step 2. Go to the line 6. To see the actual line number press `Ctrl+C`.

```
"archives": {
  "esmond_archive_1": {
    "archiver": "esmond",
    "data": {
      "measurement-agent": "{% scheduled_by_address %}",
      "url": "https://192.168.2.10/esmond/perfsonar/archive/"
    }
  }
},
```

The archiver tag *esmond_archive_1* that can be referenced in other areas of the template. The archiver type is ESnet Monitoring Daemon (*esmond*). This definition also uses the template variable `{% scheduled_by_address %}` which is replaced with an address property associated with the address object representing the agent that will schedule the task. The *url* key specify the location to store the measurement data. In this case, the data will be stored in perfSONAR2 node.

2.6 Tasks

A task is a job to do consisting of a test to be carried out, scheduling information and other options. A task in pSConfig means the same thing as a task in pScheduler. Template

variables allow pSConfig to access properties of the task components listed in the previous sections to connect the various pieces of the task together. Figure 7 shows representation of a task definition.

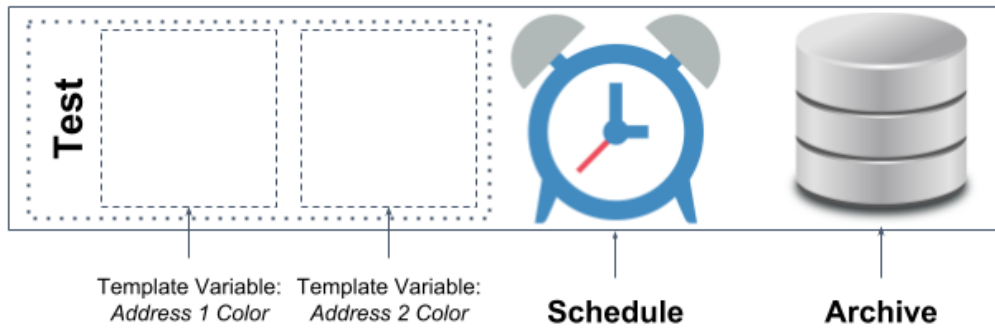


Figure 7. Task definition².

The squares represent the two addresses (source and destination) defined in a test. In this figure, those addresses are represented by colors. After the test definition, it is scheduled according to the parameters in the schedule object. Finally, the results are stored into an archiver, this feature is optional.

Step 1. In perfSONAR2 type `nano /home/template.json`, a pSConfig template will be displayed.

Step 2. Go to the line 69. To see the actual line number press `Ctrl+C`.

```

"tasks": {
  "throughput_task": {
    "group": "throughput_group",
    "test": "throughput_test",
    "schedule": "schedule_PT2M",
    "archives": [ "esmond_archive_1" ],
    "_meta": {
      "display-name": "Throughput Test"
    }
  },
  "loss_task": {
    "group": "loss_group",
    "test": "loss_test",
    "schedule": "schedule_PT2M",
    "archives": [ "esmond_archive_1" ],
    "_meta": {
      "display-name": "Loss Test"
    }
  }
}

```

In this object, there are two tasks, one for throughput measurements tagged as *throughput_task* and another for packet loss measurements tagged as *loss_task*. The group attribute is referred to *throughput_group* and *loss_group* specified before in the group object. The schedule and archiver have the same key values for both tasks. Finally, a metadata key is specified with the name of each task, those values are identifier. At this point, the template describes the task topology and can be published to be accessed by all perfSONAR node in the topology.

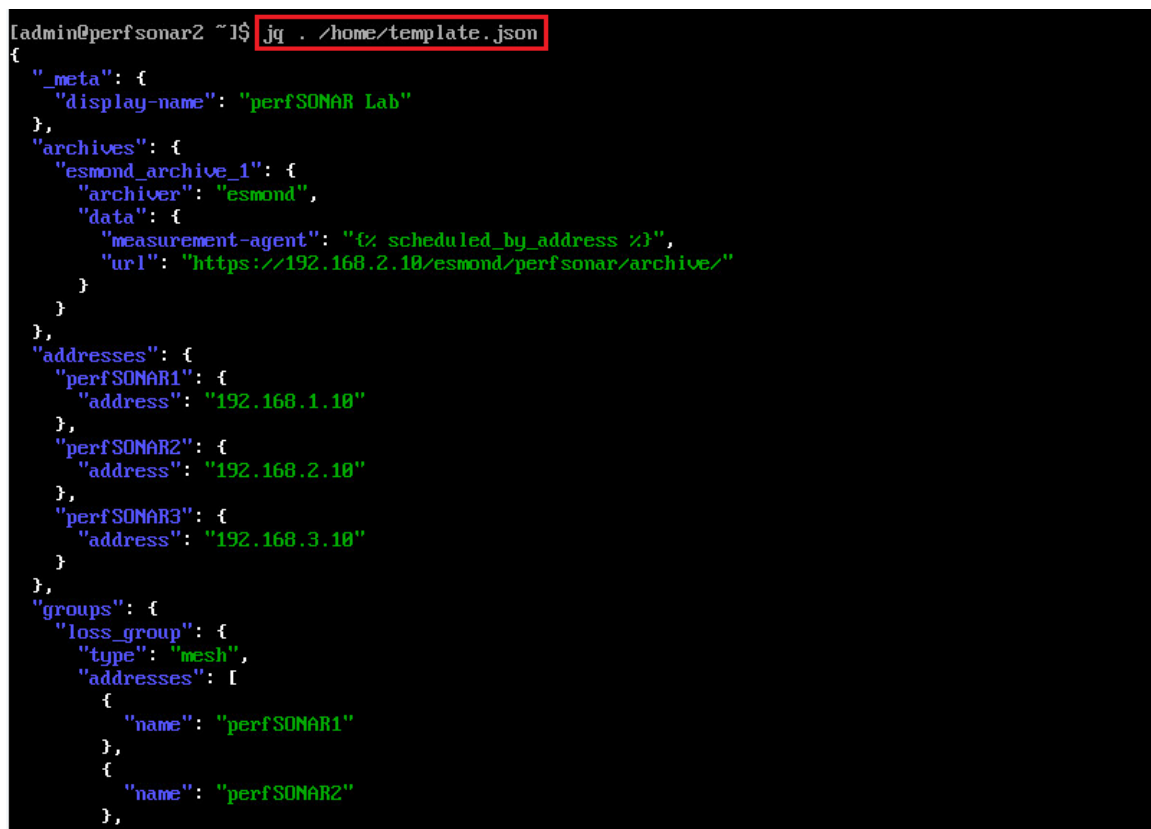
Step 3. Press `Ctrl+X` to close the window. The user could repeat the previous steps anytime along this lab

3 Publishing a pSConfig template

In this section, the user will verify the pSConfig file and the it will be published. It is necessary to publish this template to make it accessible to all perfSONAR nodes in the topology.

Step 1. To check the syntax, in perfSONAR2 command line type the command shown below. If the JSON file syntax is correct, the user will see the script on the screen.

```
jq . /home/template.json
```



```
[admin@perfsonar2 ~]# jq . /home/template.json
{
  "_meta": {
    "display-name": "perfSONAR Lab"
  },
  "archives": {
    "esmond_archive_1": {
      "archiver": "esmond",
      "data": {
        "measurement-agent": "{% scheduled_by_address %}",
        "url": "https://192.168.2.10/esmond/perfsonar/archive/"
      }
    }
  },
  "addresses": {
    "perfSONAR1": {
      "address": "192.168.1.10"
    },
    "perfSONAR2": {
      "address": "192.168.2.10"
    },
    "perfSONAR3": {
      "address": "192.168.3.10"
    }
  },
  "groups": {
    "loss_group": {
      "type": "mesh",
      "addresses": [
        {
          "name": "perfSONAR1"
        },
        {
          "name": "perfSONAR2"
        }
      ]
    }
  }
}
```

Step 2. To publish the file, type the following command:

```
sudo psconfig publish /home/template.json
```

Type `admin` as the password, notice that the password will not be displayed while typing it.

```

[admin@perfsonar2 ~]$ sudo psconfig publish /home/template.json
[sudo] password for admin:
Success! File saved to /usr/lib/perfsonar/web-psconfig/template.json

Published file can be accessed at https://localhost/psconfig/template.json

Execute the following on a host running an agent to use this file:

    psconfig remote add "https://localhost/psconfig/template.json"

[admin@perfsonar2 ~]$
    
```

Now the template is published in <https://192.168.2.10/psconfig/template.json>, and can be accessed by perfSONAR1, perfSONAR2 and perfSONAR3 nodes.

At this point, the user has configured MaDDash server. In this section, perfSONAR1 and perfSONAR3 nodes, are going to run the pSConfig agent published on <https://192.168.2.10/psconfig/template.json>.

4 Running the pSConfig pScheduler Agent

The role of the pSConfig pScheduler Agent is to read pSConfig templates and generate a set of pScheduler tasks. The figure 8 describes this role⁵.

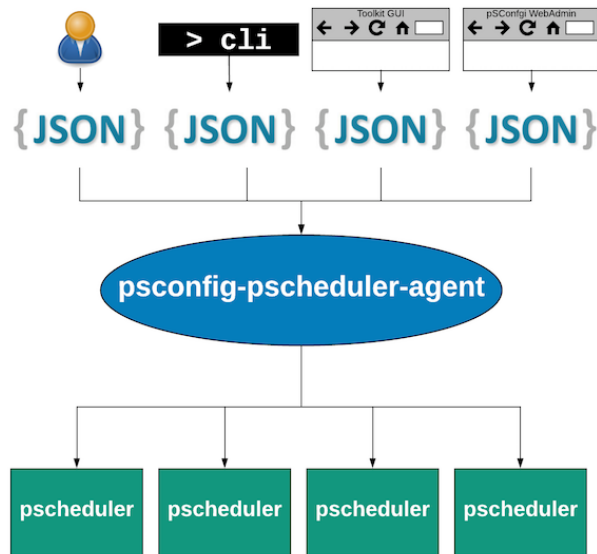


Figure 8. Workflow of the pScheduler Agent⁵.

The process depicted in the figure 8 includes the following steps:

- Read the configured templates.
- Determine the pScheduler tasks to schedule.
- Communicate with the appropriate pScheduler servers to ensure the tasks are created.

These steps are completed whenever one of the following events occur:

- The agent starts.
- The default local configuration file remains unchanged within a certain time period. By default, this value is 1 minute.
- If no changes, on a configurable interval after the start of the last run. By default, these steps are run every 1 hour.

Step 1. On the topology click on perfSONAR1 node and login typing `admin` as the username and `admin` as the password.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar1 login: admin
Password:
Last login: Wed Jan 30 15:14:47 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

  /usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar1 ~]#
```

Step 2. To run the pSConfig Agent in perfSONAR1 node, type the following command:

```
sudo psconfig remote add "https://192.168.2.10/psconfig/template.json"
```

If required, type `admin` as the password.

```
[admin@perfsonar1 ~]# sudo psconfig remote add "https://192.168.2.10/psconfig/template.json"
[sudo] password for admin:
=== pScheduler Agent ===
Replaced existing remote configuration for https://192.168.2.10/psconfig/template.json
[admin@perfsonar1 ~]#
```

At this point the pSconfig template is run by a pScheduler agent. Notice that the pSConfig template is published in perfSONAR2 node and the task are running in perfSONAR1.

5 Viewing Scheduled Tasks

Step 1. In order to visualize the tasks defined in the pSConfig template, on perfSONAR1 command line type `pscheduler monitor` to visualize the ongoing tests.

```
[admin@perfsonar1 ~]# pscheduler monitor
```

The user will see a screen with the schedule of tests. These tests have a status depending on whether they have already run or are still waiting to do so. Possible status values are:

- *Pending*: This run is scheduled to execute at some point in the future.
- *On Deck*: This run is scheduled to execute and will begin execution very soon.
- *Running*: This run is in the middle of execution.
- *Cleanup*: This run completed execution and is doing some final operations.
- *Finished*: The run has already executed and finished successfully,
- *Overdue*: The run was scheduled to execute at a certain time in the past but did not. It may get executed soon if it is not beyond a certain threshold.
- *Missed*: The run was scheduled but did not execute at its given time. This can happen if the scheduler was not running at the allotted time or the task was paused.
- *Failed*: The run failed to complete for some reason.
- *Non-Starter*: The run could not be scheduled because there were no timeslots that could accommodate the constraints.
- *Canceled*: The task was cancelled before the run was executed.

2019-04-23T19:21:18-04:00		pScheduler Monitor		perfsnar1
2019-04-23T23:18:07Z	Finished	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:18:03Z	Finished	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:18:40Z	Finished	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:19:10Z	Finished	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:20:45Z	Finished	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:21:02Z	Running	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:21:37Z	On Deck	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:21:58Z	On Deck	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:22:06Z	On Deck	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:22:38Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:23:07Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:23:50Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:24:07Z	Pending	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:24:48Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:25:20Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:25:55Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:26:15Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:26:45Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:27:08Z	Pending	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:28:26Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:28:34Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:28:46Z	Pending	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:29:52Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:30:16Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:30:23Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:31:09Z	Pending	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:31:14Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:31:21Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:32:01Z	Pending	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:32:16Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:33:17Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:34:12Z	Pending	rtt	--dest 192.168.3.10 --source 192.168.1.10	
2019-04-23T23:34:24Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	
2019-04-23T23:34:49Z	Pending	throughput	--duration PT10S --source 192.168.1.10 --dest 192.168.1.10	
2019-04-23T23:35:08Z	Pending	rtt	--dest 192.168.2.10 --source 192.168.1.10	

The results above indicate that several round-trip time test *rtt* and throughput tests are running approximately each 2 minutes. The user will see the status, the source and the destination IP addresses. In addition, the duration of the *rtt* test is shown besides it.

Step 2. To exit from pScheduler monitor, press `Ctrl+c`.

This concludes Lab 7.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swany and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 8: perfSONAR Monitoring and Debugging Dashboard

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	3
Lab roadmap.....	4
1 Introduction.....	4
2 Configuring MaDDash server.....	5
2.1 Allow http and https traffic to MaDDash server	5
2.2 Publishing pSConfig agent.....	6
2.3 Configuring central measurement archive	7
2.4 Running pSConfig MaDDash agent	10
3 Configuring perfSONAR nodes.....	11
4 Checking the grids	12
5 Visualizing the dashboard.....	15
6 Adding packet loss to interface connecting to network 192.168.2.0/24	17
References.....	22

Overview

This lab presents the perfSONAR Monitoring and Debugging Dashboard (MaDDash). This tool is aimed to collect large amounts of measurement data and display them in a two-dimensional grid referred to as a dashboard.

Objectives

By the end of this lab, the user will:

- 1 Configure MaDDash in order to visualize regular tests.
- 2 Run pSConfig agents on perfSONAR nodes.
- 3 Configure a central measurement archive.
- 4 Check the grids using MaDDash administrator web interface.
- 5 Visualize the measurement data on the dashboard.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

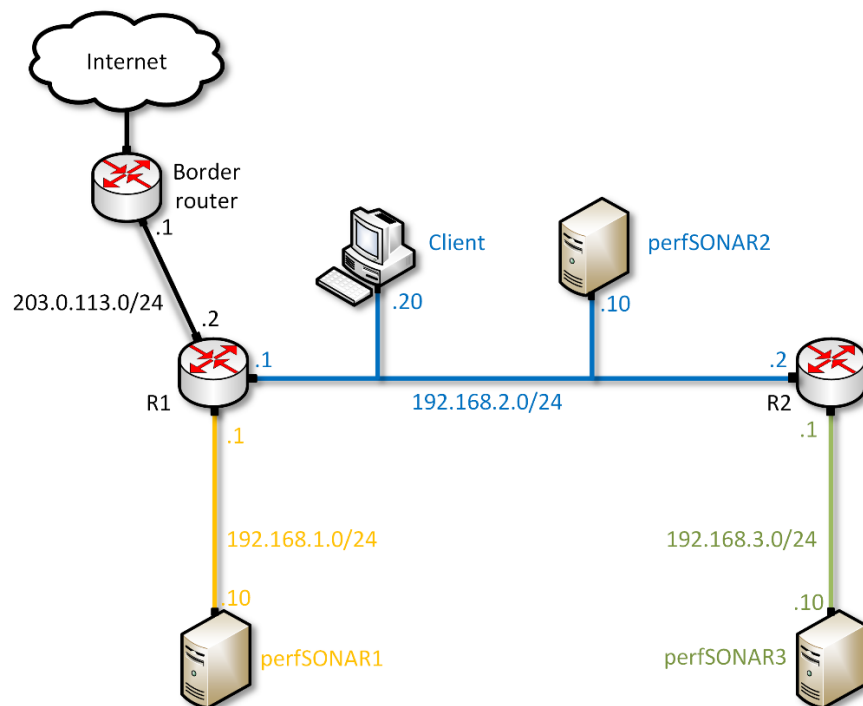


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes and the Client host.

Table 1. Credentials to access perfSONAR1, perfSONAR2, perfSONAR3 and Client.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

This lab includes the following tasks:

1. Section 1: Introduction.
2. Section 2: Configuring MaDDash server.
3. Section 3: Configuring perfSONAR nodes.
4. Section 4: Checking the grids.
5. Section 5: Visualizing the dashboard.
6. Section 6: Adding packet loss to interface connecting to network 192.168.2.0/24.

1 Introduction

In scientific collaboration environments, computational and instrumentational resources are often physically distributed over the Wide Area Networks (WAN). Collaborators work remotely in different centers and require access to instruments. As large number of data flows move between centers, data visualization becomes an important tool in facilitating network monitoring.

The Monitoring and Debugging Dashboard (MaDDash) is a software package for perfSONAR. It collects and presents two-dimensional monitoring data as a set of grids referred to as a dashboard. Many monitoring systems emphasize one-dimensional graphs, however network managers can face difficulties presenting the measurement data as complexity increases. Therefore, MaDDash provides the tools to create, configure and synchronize tests which are running on multiple hosts. These results can then be accessed using a REST API which provides the building blocks for components such as the included web interface that presents the data as a set of grids¹.

MaDDash is a part of the visualization layer, as shown in the figure 2. It can run a pSConfig template and can retrieve the measurement data of other perfSONAR nodes through the ESnet Monitoring Daemon (Esmond), which is a system for collecting, storing, visualizing and analyzing large sets of timeseries data.

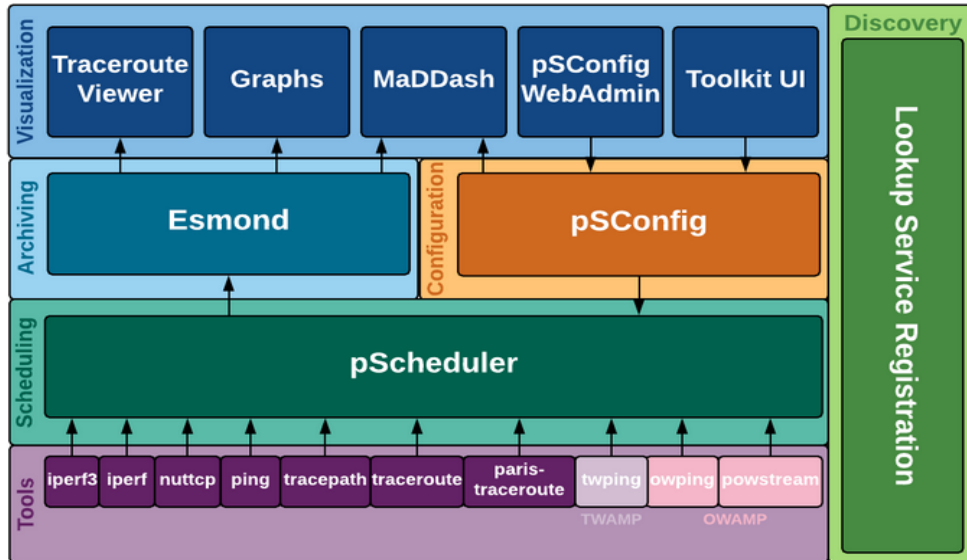


Figure 2. perfSONAR layers².

2 Configuring MaDDash server

In this section the user will configure a MaDDash server in order run and collect measurement data. First, the user will open firewall to allow http/https traffic. Second, the user will publish a pSConfig template to run the measurement tests in the server and the other nodes. Then, it is shown how to configure the central measurement archive. Finally, the user will run a pSConfig MaDDash agent.

2.1 Allow http and https traffic to MaDDash server

In order to provide access the server, it is necessary to open the firewall to allow http/https traffic.

Step 1. Login to perfSONAR2 typing the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar2 login: admin
Password:
Last login: Sun Jan 27 17:27:33 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

    /usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
'abrt-cli status' timed out
[admin@perfsonar2 ~]$_
```

Step 2. To allow traffic through the http port (80) type the command shown below. The user will be required to enter the password as `admin`. Notice that the password will not be displayed while typing it.

```
sudo firewall-cmd --permanent --add-port=80/tcp
```

```
[admin@perfsonar2 ~]$_ sudo firewall-cmd --permanent --add-port=80/tcp
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.

[sudo] password for admin:
success
[admin@perfsonar2 ~]$_
```

Step 3. Similarly, to allow traffic through the http port (443) type the command shown below.

```
sudo firewall-cmd --permanent --add-port=443/tcp
```

```
[admin@perfsonar2 ~]$_ sudo firewall-cmd --permanent --add-port=443/tcp
[sudo] password for admin:
success
[admin@perfsonar2 ~]$_ firewall-cmd --reload
Authorization failed.
Make sure polkit agent is running or run the application as superuser.
[admin@perfsonar2 ~]$_ sudo firewall-cmd --reload
success
[admin@perfsonar2 ~]$_
```

Step 4. To apply the changes, type the following command:

```
sudo firewall-cmd --reload
```

```
[admin@perfsonar2 ~]$_ sudo firewall-cmd --reload
success
[admin@perfsonar2 ~]$_
```

2.2 Publishing pSConfig agent

In this lab, the user is provided a template. This template is a pSConfig archive which runs pScheduler tasks. It is necessary to publish this template to make it accessible to all perfSONAR nodes.

Step 1. In the directory `/home/`, the user will find the pSConfig agent `template.json`, which runs pScheduler tasks. To check for errors, type the command shown below. If the JSON file syntax is correct, the user will see the script echoed on the screen.

```
jq . /home/template.json
```

```
admin@perfsonar2 ~1$ jq . /home/template.json
{
  "name": "perfSONAR3"
},
{
  "tests": {
    "test_throughput_1": {
      "type": "throughput",
      "spec": {
        "source": "% address[0] %",
        "dest": "% address[1] %",
        "duration": "PT10S"
      }
    }
  },
  "schedules": {
    "schedule_PT1M": {
      "repeat": "PT1M",
      "sliprand": true,
      "slip": "PT1M"
    }
  },
  "tasks": {
    "task_throughput_1": {
      "group": "group_mesh_1",
      "test": "test_throughput_1",
      "schedule": "schedule_PT1M",
      "archives": [
        "esmond_archive_1"
      ],
      "meta": {
        "display-name": "Throughput Test"
      }
    }
  }
}
```

Step 2. To publish the template, type the command shown below. The script will be available at <https://192.168.2.10/psconfig/template.json>.

```
sudo psconfig publish /home/template.json
```

```
admin@perfsonar2 ~1$ sudo psconfig publish /home/template.json
Success! File saved to /usr/lib/perfsonar/web-psconfig/template.json

Published file can be accessed at https://localhost/psconfig/template.json

Execute the following on a host running an agent to use this file:

psconfig remote add "https://localhost/psconfig/template.json"

admin@perfsonar2 ~1$ _
```

Now, the pSConfig template is published and can be accessed by perfSONAR1 and perfSONAR3 nodes.

2.3 Configuring central measurement archive

Each host will collect measurement data specified in the pSConfig template. This measurement data will be stored into a centralized database, the ESnet Monitoring Daemon (esmond). Esmond is a system for collecting, storing, visualizing and analyzing

large sets of timeseries data. In order to store measurement data, each host needs to authenticate to the Central Measurement Archive esmond.

To allow each node to store its measurement data, the user will use the following command format:

```
sudo /usr/sbin/esmond_manage add_user_ip_address admin <IP_ADDRESS>
```

- `sudo`: enables the execution of the command with higher security privileges.
- `/usr/sbin/esmond_manage`: is the route to the script to register a perfSONAR node.
- `add_user_ip_address`: is a script which allows a perfSONAR node to have access to the central measurement archive and store its measurement data.
- `admin`: is the administrator password.
- `<IP_ADDRESS>`: is the IP address of the perfSONAR node to be registered.

Step 1. To allow perfSONAR1 node to store its measurement data, in perfSONAR2 CLI type the command shown below. The user will be required to enter the password as `admin`. Notice that the password will not be displayed while typing it.

```
sudo /usr/sbin/esmond_manage add_user_ip_address admin 192.168.1.10
```

```
[admin@perfsonar2 ~]# sudo /usr/sbin/esmond_manage add_user_ip_address admin 192.168.1.10
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Schema check done
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [INFO] Connected to ['localhost:9160']
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Schema check done
cassandra_db [INFO] Schema check done
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [INFO] Connected to ['localhost:9160']
cassandra_db [INFO] Connected to ['localhost:9160']
User admin exists
Setting metadata POST permissions.
api | ps metadata | Can add ps metadata
api | ps metadata | Can change ps metadata
api | ps metadata | Can delete ps metadata
api | ps point to point subject | Can add ps point to point subject
api | ps point to point subject | Can change ps point to point subject
api | ps point to point subject | Can delete ps point to point subject
api | ps event types | Can add ps event types
api | ps event types | Can change ps event types
api | ps event types | Can delete ps event types
api | ps metadata parameters | Can add ps metadata parameters
api | ps metadata parameters | Can change ps metadata parameters
api | ps metadata parameters | Can delete ps metadata parameters
api | ps network element subject | Can add ps network element subject
api | ps network element subject | Can change ps network element subject
api | ps network element subject | Can delete ps network element subject
Setting timeseries permissions.
IP 192.168.1.10/32 already assigned to admin, skipping creation
[admin@perfsonar2 ~]#
```

Step 2. To allow perfSONAR2 node to store its measurement data, in perfSONAR2 CLI type the following command:

```
sudo /usr/sbin/esmond_manage add_user_ip_address admin 192.168.2.10
```

```

[admin@perfsonar2 ~]# sudo /usr/sbin/esmond_manage add_user_ip_address admin 192.168.2.10
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Schema check done
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [INFO] Connected to ['localhost:9160']
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Schema check done
cassandra_db [INFO] Schema check done
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [INFO] Connected to ['localhost:9160']
cassandra_db [INFO] Connected to ['localhost:9160']
User admin exists
Setting metadata POST permissions.
api | ps metadata | Can add ps metadata
api | ps metadata | Can change ps metadata
api | ps metadata | Can delete ps metadata
api | ps point to point subject | Can add ps point to point subject
api | ps point to point subject | Can change ps point to point subject
api | ps point to point subject | Can delete ps point to point subject
api | ps event types | Can add ps event types
api | ps event types | Can change ps event types
api | ps event types | Can delete ps event types
api | ps metadata parameters | Can add ps metadata parameters
api | ps metadata parameters | Can change ps metadata parameters
api | ps metadata parameters | Can delete ps metadata parameters
api | ps network element subject | Can add ps network element subject
api | ps network element subject | Can change ps network element subject
api | ps network element subject | Can delete ps network element subject
Setting timeseries permissions.
Creating entry for IP 192.168.2.10 belonging to admin
[admin@perfsonar2 ~]# _

```

Step 3. To allow perfSONAR3 node to store its measurement data, in perfSONAR2 CLI type the following command:

```
sudo /usr/sbin/esmond_manage add_user_ip_address admin 192.168.3.10
```

```

[admin@perfsonar2 ~]# sudo /usr/sbin/esmond_manage add_user_ip_address admin 192.168.3.10
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Schema check done
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [INFO] Connected to ['localhost:9160']
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Checking/creating column families
cassandra_db [INFO] Schema check done
cassandra_db [INFO] Schema check done
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [DEBUG] Opening ConnectionPool
cassandra_db [INFO] Connected to ['localhost:9160']
cassandra_db [INFO] Connected to ['localhost:9160']
User admin exists
Setting metadata POST permissions.
api | ps metadata | Can add ps metadata
api | ps metadata | Can change ps metadata
api | ps metadata | Can delete ps metadata
api | ps point to point subject | Can add ps point to point subject
api | ps point to point subject | Can change ps point to point subject
api | ps point to point subject | Can delete ps point to point subject
api | ps event types | Can add ps event types
api | ps event types | Can change ps event types
api | ps event types | Can delete ps event types
api | ps metadata parameters | Can add ps metadata parameters
api | ps metadata parameters | Can change ps metadata parameters
api | ps metadata parameters | Can delete ps metadata parameters
api | ps network element subject | Can add ps network element subject
api | ps network element subject | Can change ps network element subject
api | ps network element subject | Can delete ps network element subject
Setting timeseries permissions.
Creating entry for IP 192.168.3.10 belonging to admin
[admin@perfsonar2 ~]#

```

2.4 Running pSConfig MaDDash agent

The role of the pSConfig MaDDash agent is to read pSConfig templates and generate a set of grids to be displayed by MaDDash.

Step 1. In perfSONAR2 command line type the command shown below to publish the MaDDash agent at the given URL.

```
sudo psconfig remote add "https://192.168.2.10/psconfig/template.json"
```

```
[admin@perfsonar2 ~]# sudo psconfig remote add "https://192.168.2.10/psconfig/template.json"
=== pScheduler Agent ===
Added remote configuration https://192.168.2.10/psconfig/template.json
=== MaDDash Agent ===
Added remote configuration https://192.168.2.10/psconfig/template.json
[admin@perfsonar2 ~]#
```

Step 2. Now, perfSONAR2 node is collecting measurement data specified in the pSConfig template. To proceed, the user will restart Apache Cassandra database typing the command shown below:

```
sudo systemctl restart cassandra
```

```
[admin@perfsonar2 ~]# sudo systemctl restart cassandra
[admin@perfsonar2 ~]#
```

Step 3. In order to restart MaDDash server, type the following command:

```
sudo systemctl restart maddash-server
```

```
[admin@perfsonar2 ~]# sudo systemctl restart maddash-server
[admin@perfsonar2 ~]#
```

Step 4. To restart MaDDash agent, type the following command:

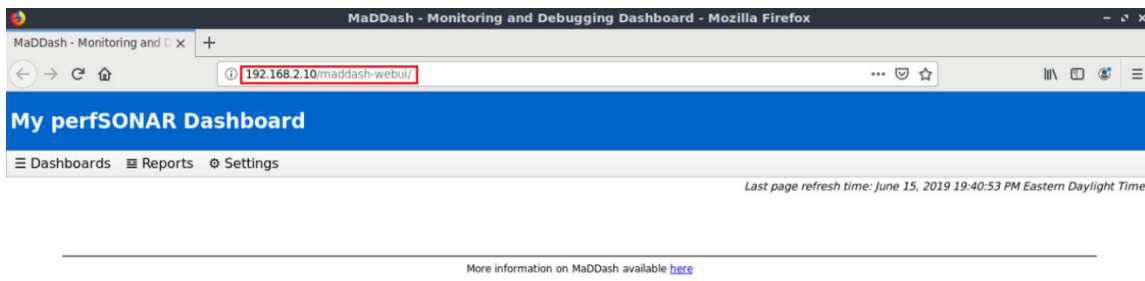
```
sudo systemctl restart psconfig-maddash-agent
```

```
[admin@perfsonar2 ~]# sudo systemctl restart psconfig-maddash-agent
[admin@perfsonar2 ~]#
```

Step 5. At this point MaDDash web interface is set up and running. To check it, go to the topology and login to the Client host and open the web browser.



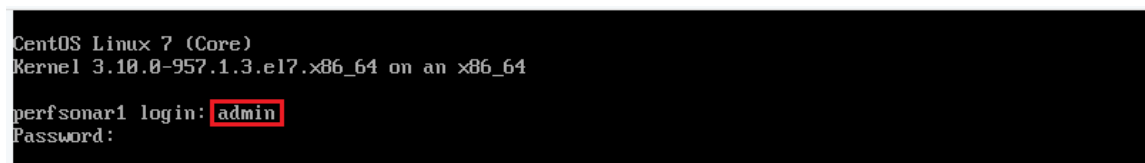
Step 7. In order to access to the dashboard, type the following URL *https://192.168.2.10/maddash-webui/*. If the web server is running, the user will see the MaDDash web user interface.



3 Configuring perfSONAR nodes

At this point, the MaDDash server has been configured. In this section, the user will configure perfSONAR1 and perfSONAR3 nodes in order to run the pSConfig agent published on *https://192.168.2.10/psconfig/template.json*.

Step 1. On the topology click on perfSONAR1 node and login typing `admin` as the username and `admin` as the password.



Step 2. To add a remote pSConfig template on perfSONAR1, type the command shown below. The user will be required to enter the password as `admin`. Notice that the password will not be displayed while typing it.

```
sudo psconfig remote add --configure-archives
"https://192.168.2.10/psconfig/template.json"
```

```
[admin@perfsonar1 ~]# sudo psconfig remote add --configure-archives "https://192.168.2.10/psconfig/t
emplate.json"
[sudo] password for admin:
=== pScheduler Agent ===
Added remote configuration https://192.168.2.10/psconfig/template.json
[admin@perfsonar1 ~]# _
```

Step 3. On the topology, click on perfSONAR3 node and login with typing `admin` as the username and `admin` as the password.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64
perfsonar3 login: admin
Password: _
```

Step 4. To add a remote pSConfig template on perfSONAR3 type the command shown below. The user will be required to enter the password as `admin`. Notice that the password will not be displayed while typing it.

```
sudo psconfig remote add --configure-archives
"https://192.168.2.10/psconfig/template.json"
```

```
[admin@perfsonar3 ~]# sudo psconfig remote add --configure-archives "https://192.168.2.10/psconfig/t
emplate.json"
[sudo] password for admin:
=== pScheduler Agent ===
Added remote configuration https://192.168.2.10/psconfig/template.json
[admin@perfsonar3 ~]# _
```

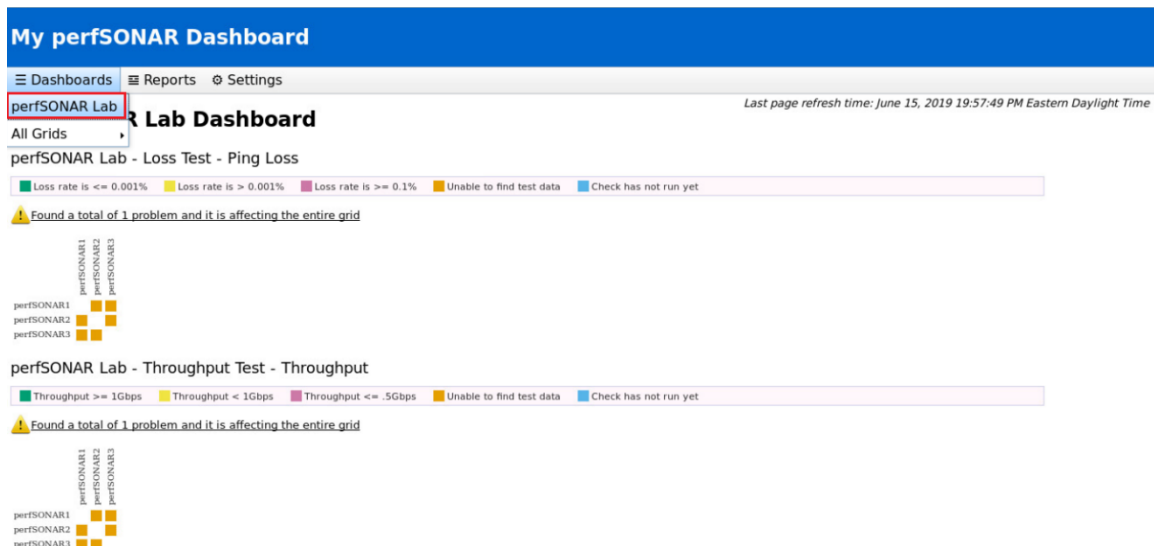
4 Checking the grids

At this point, the MaDDash server is running and collecting data. It takes time to have the measurement data propagated and displayed on the grid. To avoid waiting, the user will access the MaDDash Administrator Web Interface. The administrator web interface allows privileged users to perform special operations on the dashboard. The allowed operations are:

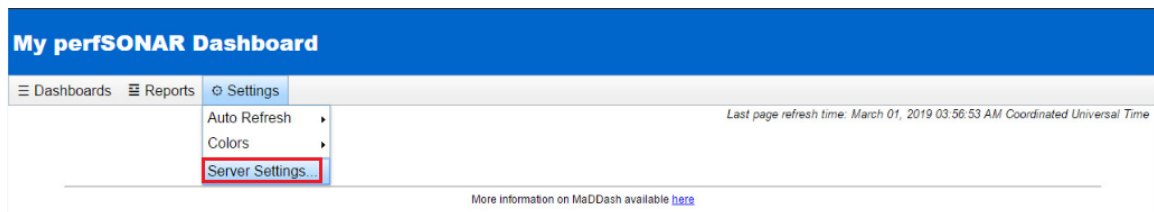
- Re-scheduling a check to run at a certain time.
- Scheduling an event, such as a maintenance window, that may impact check results.
- Viewing and canceling existing events.

The steps that follow describe how to reschedule a check event. This check event will propagate measurement data on the dashboard immediately.

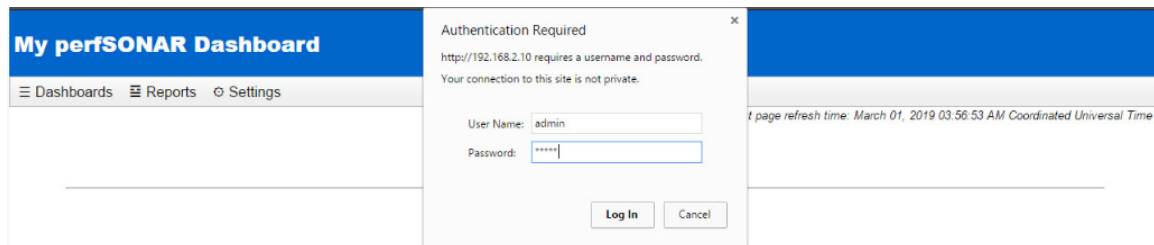
Step 1. On the Client host, click on *Dashboards > perfSONAR Lab*. The user will see the dashboard, but in this case the measurement data is not displayed because by default it needs time to be propagated and visualized on the dashboard.



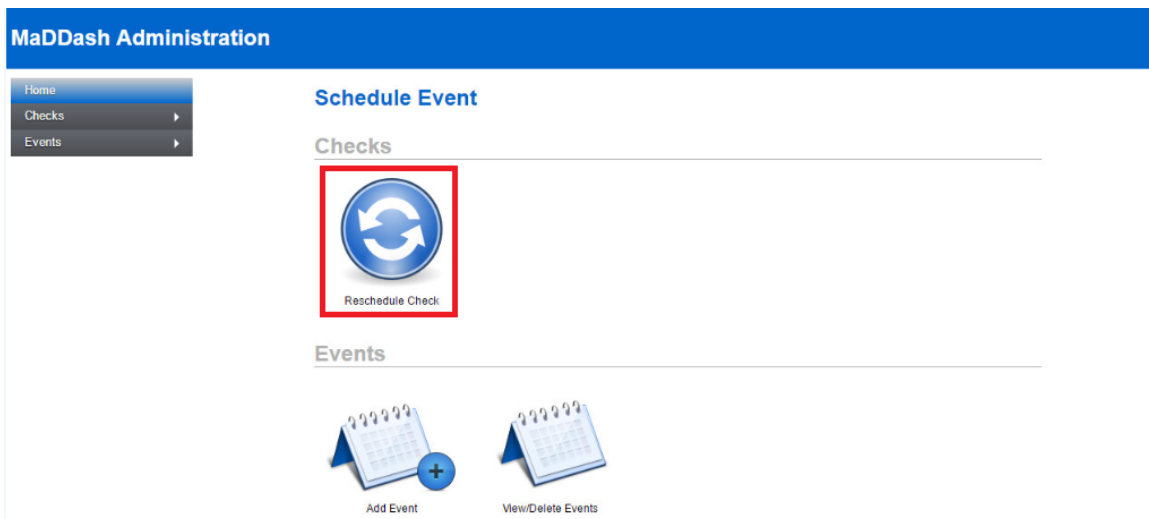
Step 2. To display the measurement data on the dashboard immediately, the user will modify the Server Settings to schedule a Check. To proceed, click on *Settings > Server Settings*.



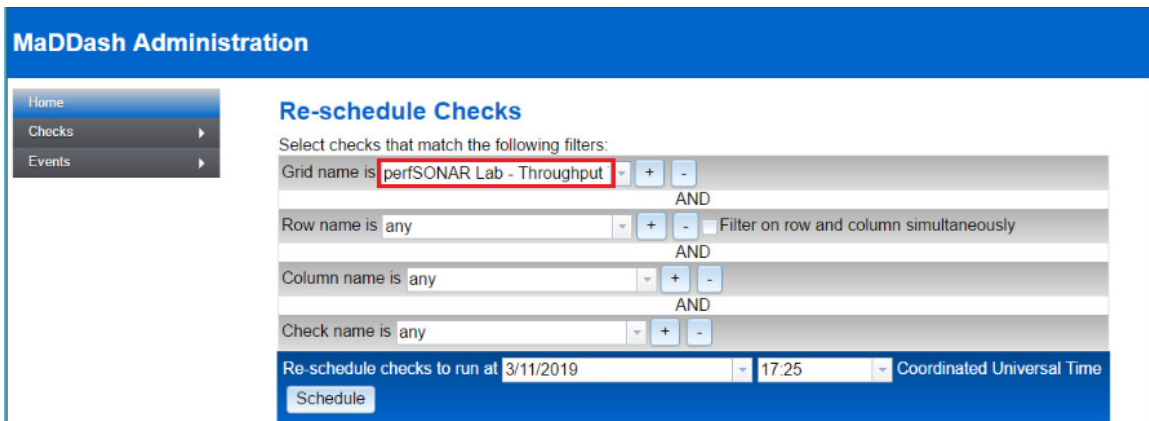
Step 3. The user will be required to authenticate. To proceed type username `admin` and password `admin`.



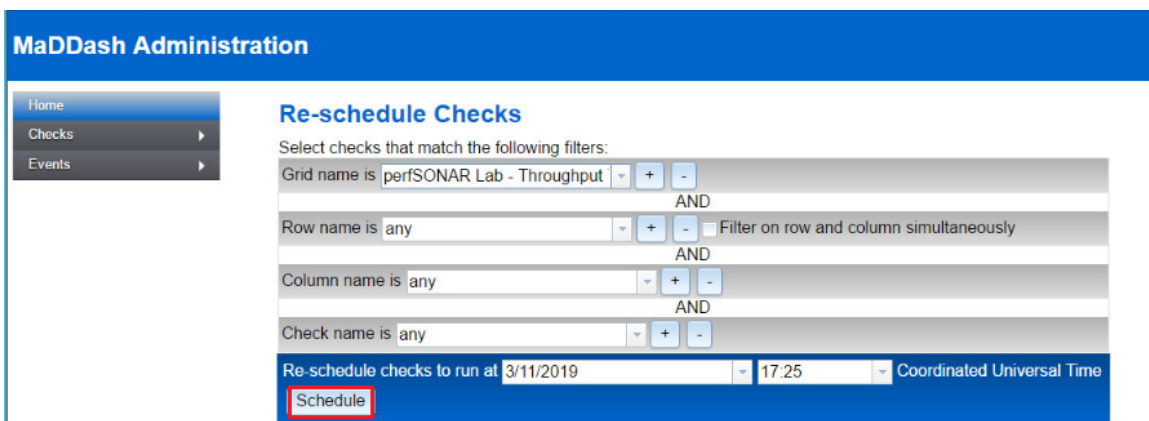
Step 4. The user will see the Administrator Web Interface. Click on Reschedule Check. This will force the dashboard to show the measurement data as soon as possible.



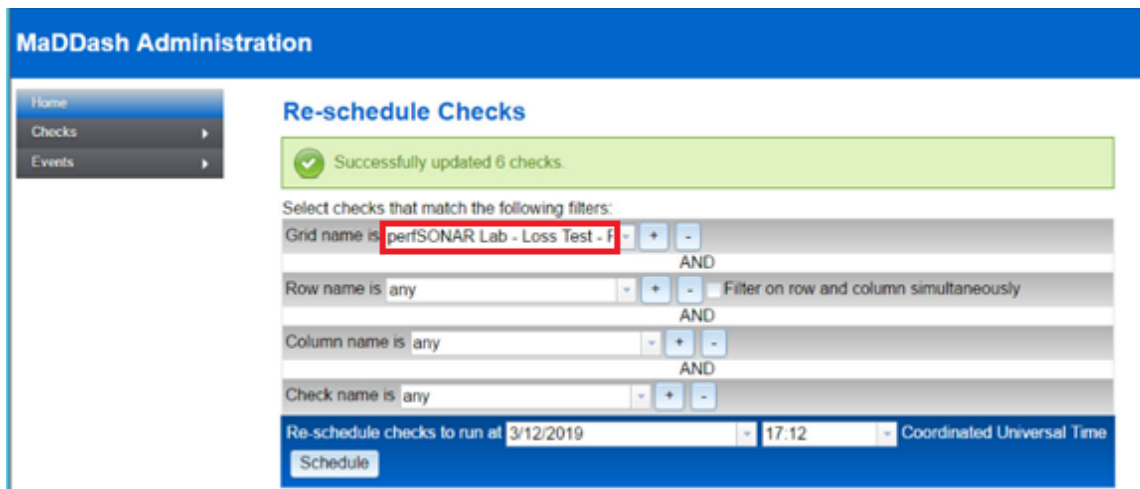
Step 5. Select the grid name as *perfSONAR Lab – Throughput*.



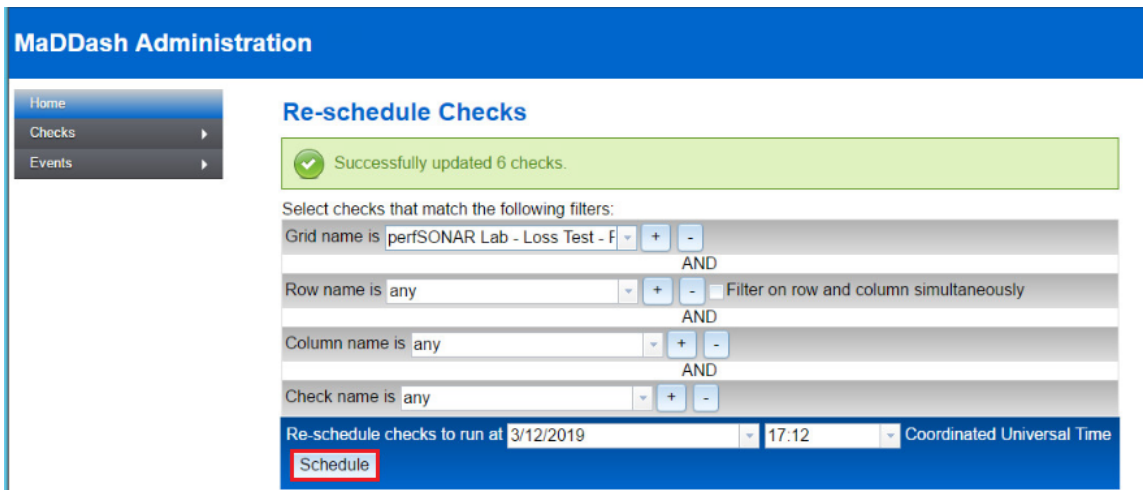
Step 6. To apply the configuration, click on *Schedule*.



Step 7. Select the grid name as *perfSONAR Lab – Loss Test*.

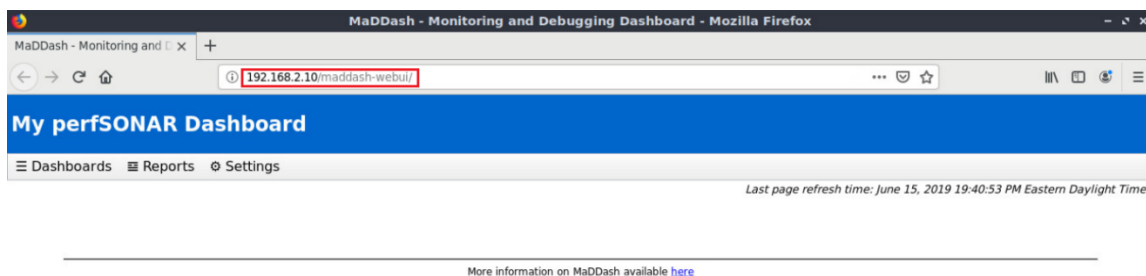


Step 8. To apply the configuration, click on *Schedule*.

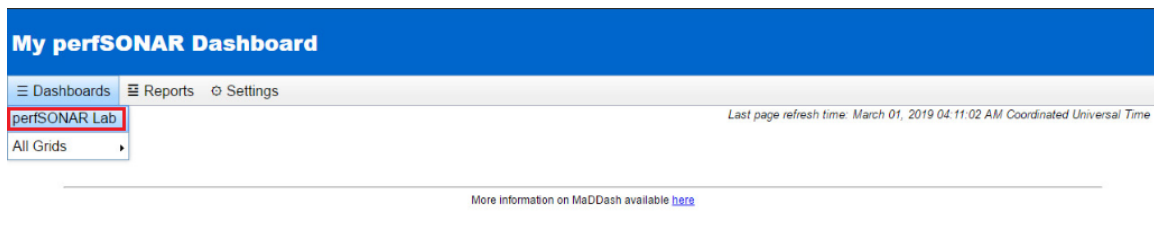


5 Visualizing the dashboard

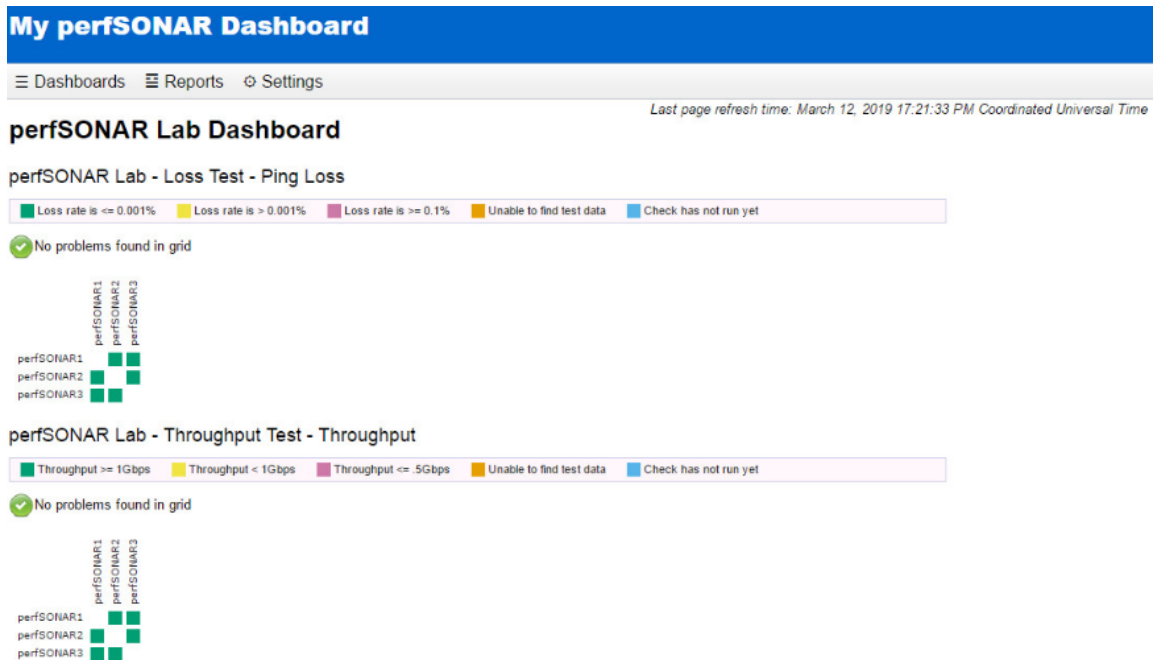
Step 1. In order to access the dashboard, type the following URL <https://192.168.2.10/maddash-webui/>. If the web server is running, the user will see MaDDash web user interface.



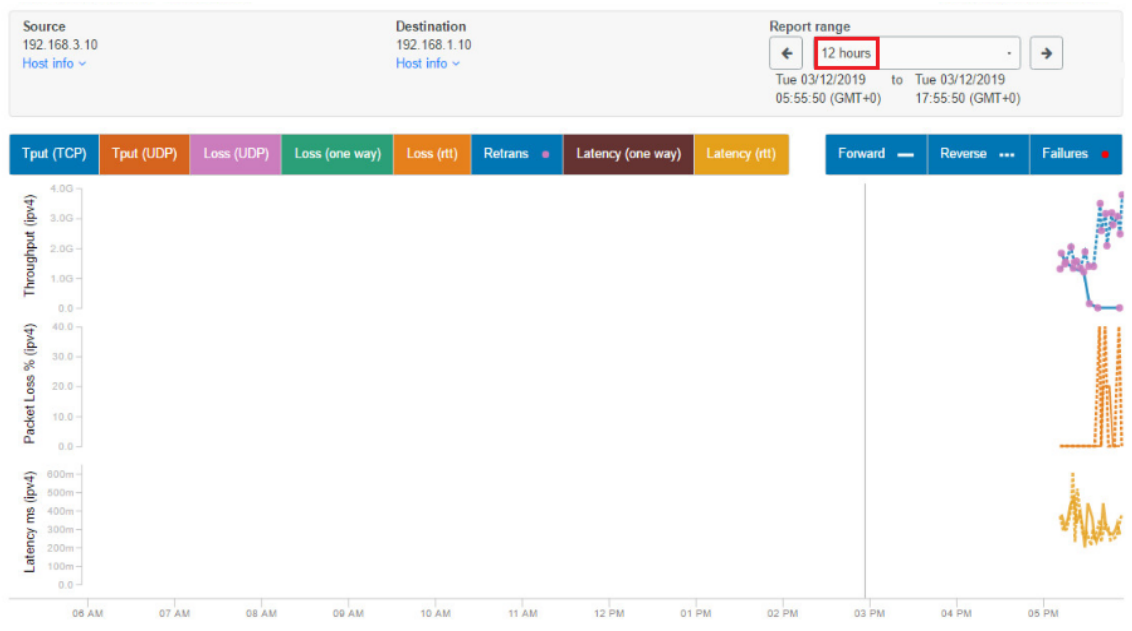
Step 2. In the Web User Interface select *Dashboard > perfSONAR Lab*



Step 3. The user will see the Throughput and Loss Dashboards which are from perfSONAR1, perfSONAR2 and perfSONAR3 nodes. To visualize the results in a timing graph, the user can click on any green square and the browser will open a new tab. On the Throughput dashboard, click on square located on first row and third column to visualize the results of the throughput test between perfSONAR1 and perfSONAR3 nodes.



Step 4. In this graph the results collected since the pScheduler agents started are shown, and the user can visualize the throughput when the source is 192.168.1.10 and the destination is 192.168.3.10. To adjust the time range, select an appropriate value on Report range to see the results with more detail. The throughput graph shows bidirectional throughput, failures and retransmissions on the same plot. Below, the timing graph of the packet loss and latency are displayed.



6 Adding packet loss to interface connecting to network 192.168.2.0/24

In this section, the user will add a 10% packet loss to the router R1 and router R2 using Network Emulator (NETEM) commands. This change will affect the performance of the network. The user will see the effects on the dashboard.

Step 1. Open router R2 and enter the username `root` and password `password`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 4.19.1-1.el7.elrepo.x86_64 on an x86_64
R2 login: root
Password:
```

Step 2. Identify the interfaces which are connected to the network 192.168.2.0/24 on router R2. In router R2 command line, type the command `ifconfig`. This command displays information related to the network interfaces in the local device.

```
[root@R2 ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.1 netmask 255.255.255.0 broadcast 192.168.3.255
    inet6 fe80::250:56ff:feae:e5dc prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:e5:dc txqueuelen 1000 (Ethernet)
    RX packets 1013392857 bytes 1488074592733 (1.3 TiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1217711915 bytes 2909379957450 (2.6 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.2 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::10a6:2962:4b7e:c21a prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:ae:96:6a txqueuelen 1000 (Ethernet)
    RX packets 1213137503 bytes 1799533693195 (1.6 TiB)
    RX errors 0 dropped 10 overruns 0 frame 0
    TX packets 1016660699 bytes 2402663209617 (2.1 TiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 468 bytes 37528 (36.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 468 bytes 37528 (36.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@R2 ~]#
```

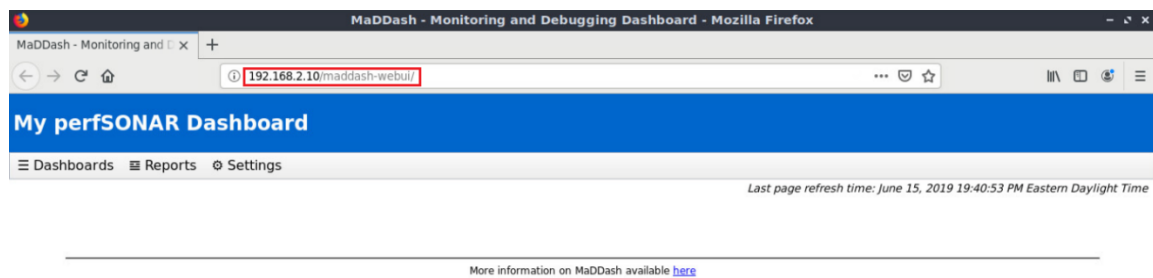
Notice that the interface *ens37* is connected to the network *192.168.2.0/24*.

Step 3. In order to add a 10% packet loss, `sudo` in router R2 command line, type the following command:

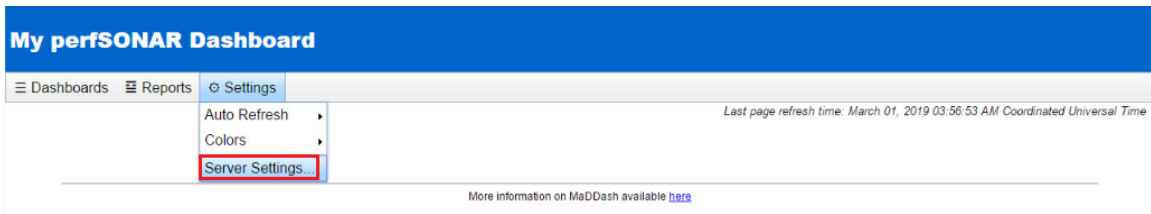
```
sudo tc qdisc add dev ens37 root netem loss 10%
```

```
[root@R2 ~]# sudo tc qdisc add dev ens37 root netem loss 10%
[root@R2 ~]#
```

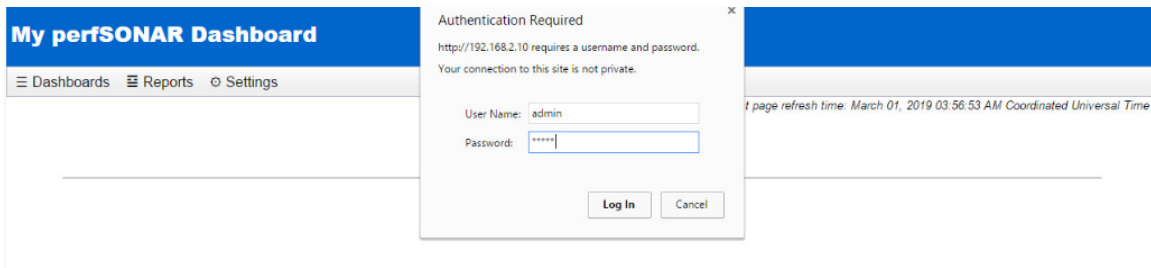
Step 4. Wait for at least 2 minutes to get the data propagated to the dashboard, then go to the administrator web interface. In order to access the dashboard, type the following URL <https://192.168.2.10/maddash-webui/>.



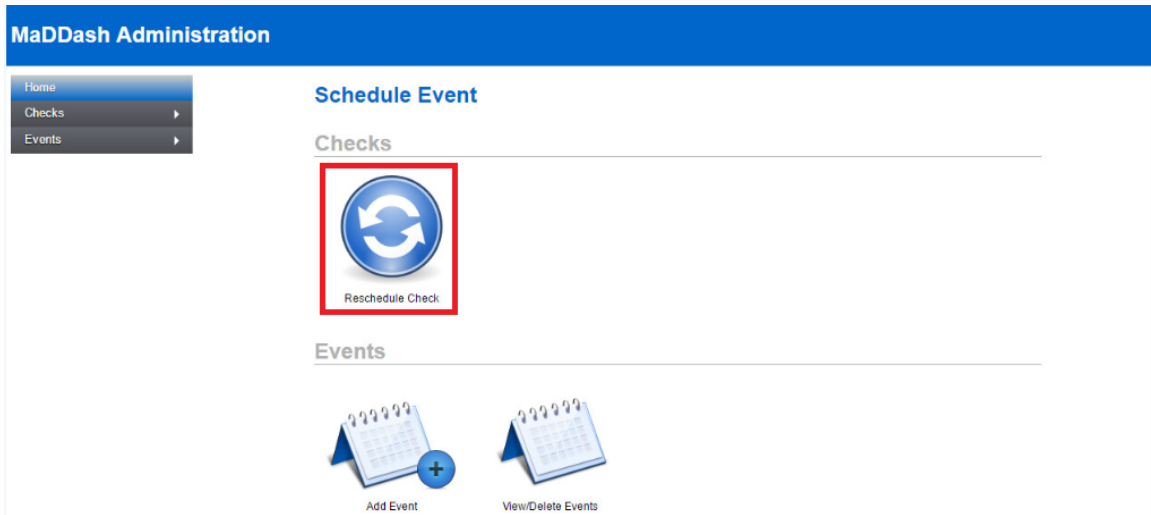
Step 5. In the Web User Interface select *Settings > Server Settings*.



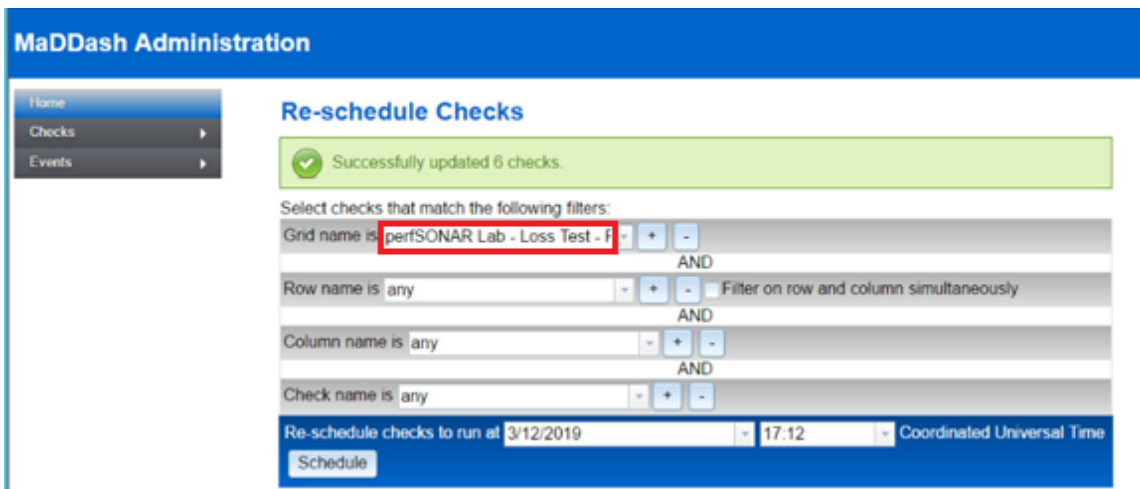
Step 6. The user may be required to authenticate. To proceed type username `admin` and password `admin`.



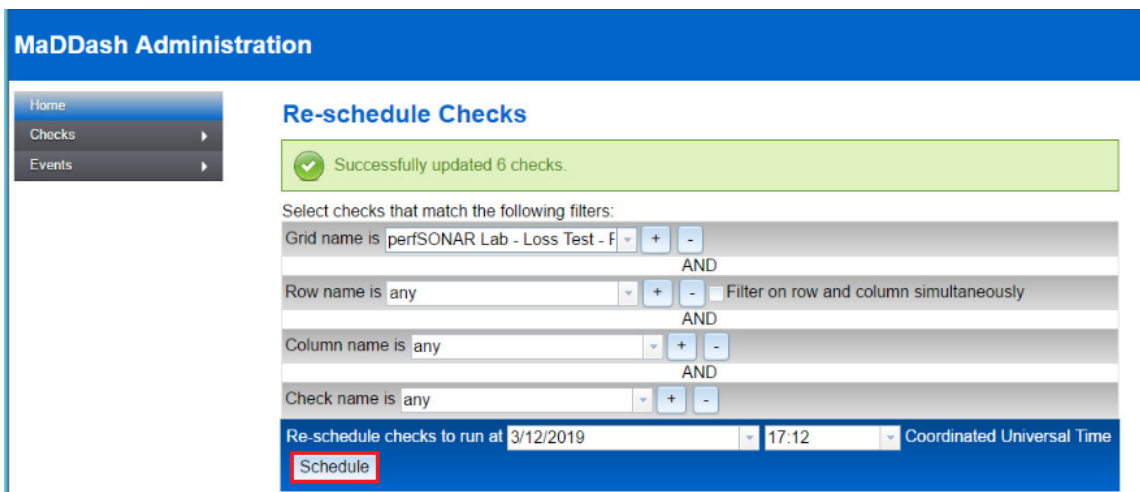
Step 7. The user will see the Administrator Web Interface. Click on Reschedule Check. This will force the dashboard to show the measurement data as sooner than it would otherwise.



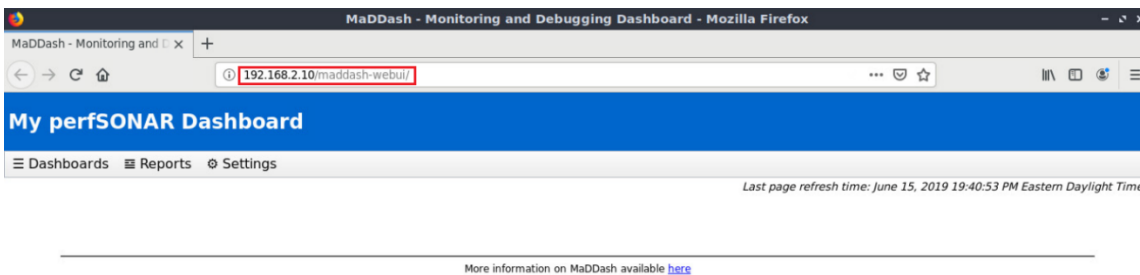
Step 8. Select the grid name as *perfSONAR Lab – Loss Test*.



Step 9. To apply the configuration, click on Schedule and wait 1 minute to enter again to MaDDash web interface <http://192.168.2.10/maddash-webui>.



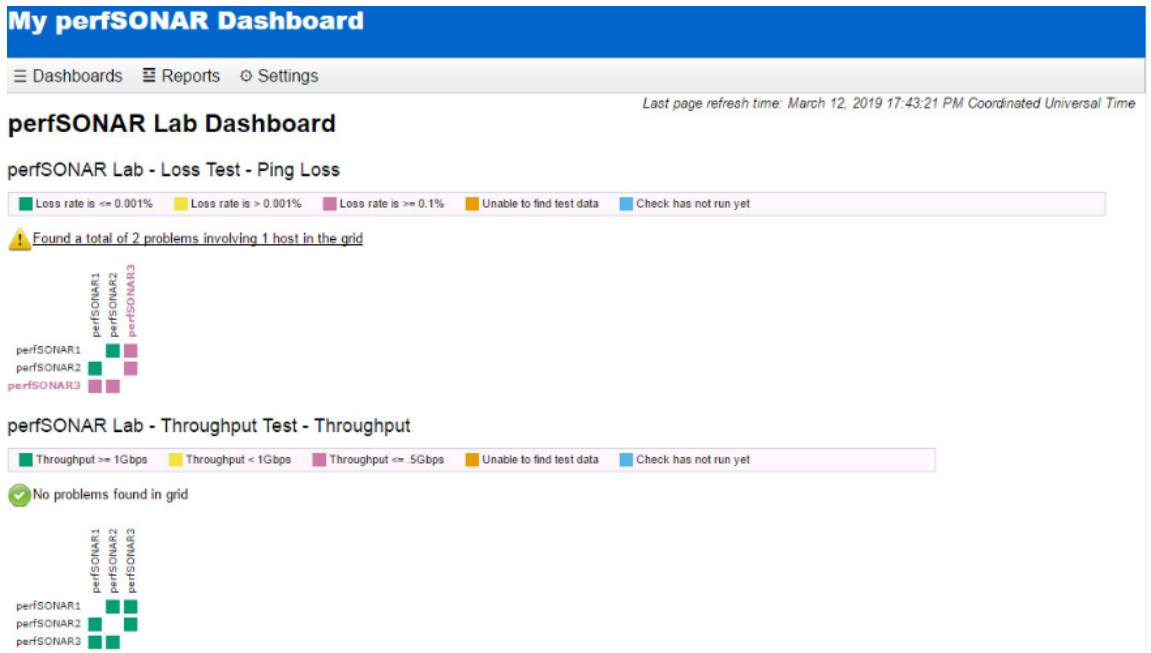
Step 10. In order to access the dashboard, type the following URL <https://192.168.2.10/maddash-webui/>. If the web server is running, the user will see MaDDash web user interface.



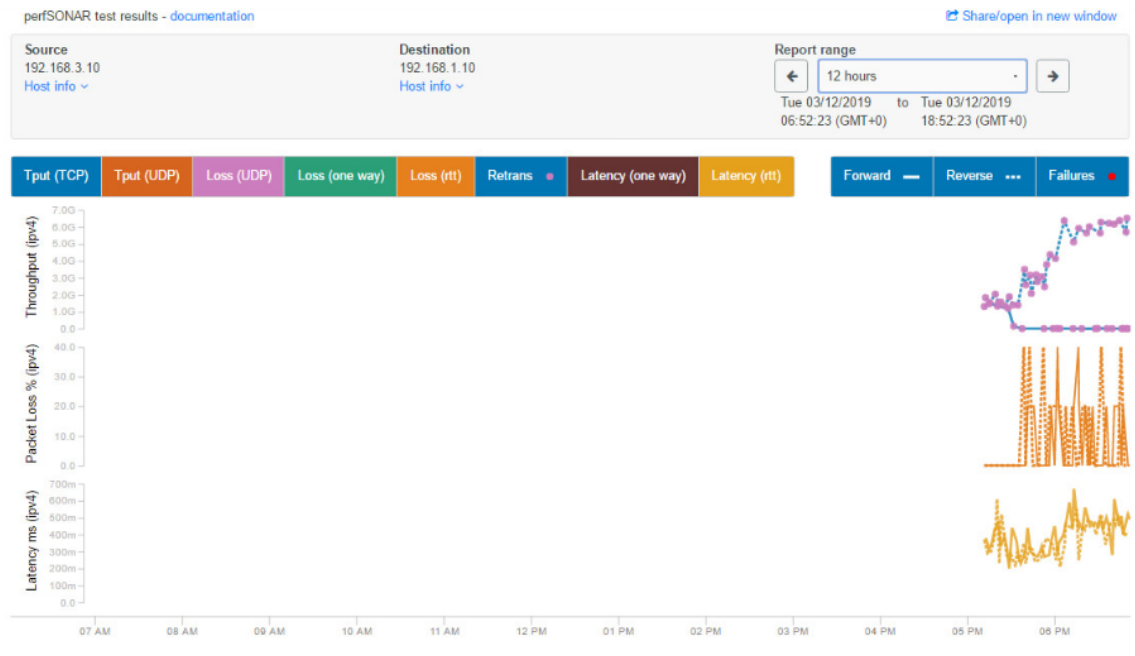
Step 11. In the Web User Interface select *Dashboard > perfSONAR Lab*



Step 12. The user will see the Throughput and Loss Dashboards which are from perfSONAR1, perfSONAR2 and perfSONAR3 nodes. Now the user will see that the loss rate between perfSONAR3 and the other nodes are affected.



Step 13. To visualize the results in a timing graph, the user can click on any green square and the browser will open a new tab. On the Loss dashboard, click on the square located on third row and first column to visualize the results of the throughput test between perfSONAR3 and perfSONAR1 nodes.



This graph shows the results collected since the pScheduler agents started, the user can visualize the throughput when the source is *192.168.3.10* and the destination is *192.168.1.10*. To adjust the time range, select an appropriate value on Report range to see the results with more detail. Bidirectional throughput, failures and retransmissions are shown on the same plot.

This concludes Lab 8.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swany and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 9: pSConfig Web Administrator

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings	4
Lab roadmap	4
1 Introduction	4
1.1 PWA overview	5
2 Configuring hosts	6
2.1 Accessing the web user interface	6
2.2 Adding hosts to the directory	7
3 Configuring host groups	11
3.1 Configuring Throughput Group	11
3.2 Configuring latency group	14
4 Setting test specifications	16
4.1 Configuring throughput test specification	17
4.2 Configuring latency test specification	19
5 Creating pSConfig output	22
5.1 Adding throughput test	24
5.2 Adding latency test	27
6 Visualizing the measurement data using pScheduler monitor	30
References	31

Overview

This lab presents how to create and publish pSConfig templates using pSConfig Web Administrator (PWA). This tool is a web-based user interface for perfSONAR administrators to define and publish pSConfig templates, which automates tests executed by test nodes, and provides topology information to various services, such as MadDash.

Objectives

By the end of this lab, the user will:

1. Understand PWA architecture.
2. Create host groups.
3. Define tests specifications.
4. Configure test parameters.
5. Publish pSConfig archive.
6. Run pSConfig pScheduler Agent.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

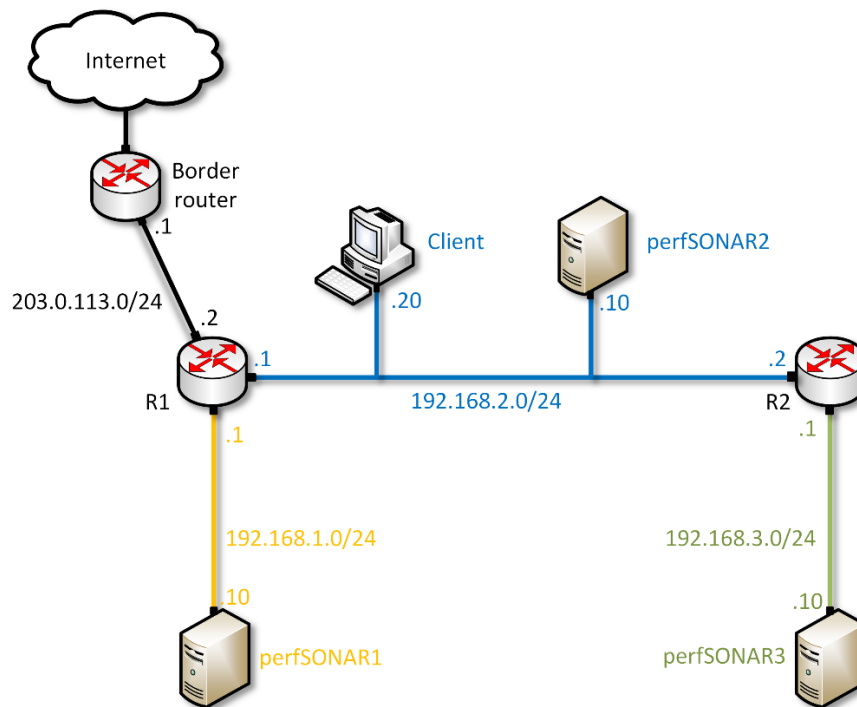


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes and the Client host.

Table 1. Credentials to access perfSONAR1, perfSONAR2, perfSONAR3 and Client.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

This lab includes the following tasks:

1. Section 1: Introduction.
2. Section 2: Configuring hosts.
3. Section 3: Configuring host groups.
4. Section 4: Setting test specifications.
5. Section 5: Configuring pSConfig output.
6. Section 6: Visualizing the measurement data using pScheduler monitor.

1 Introduction

pSConfig Web Administrator (PWA) is a web-based user interface for perfSONAR administrators to define and publish pSConfig configuration files. The output automates tests executed by test nodes, and provides topology information to various services, such as MaDDash.

In addition to providing a user-friendly interface for creating pSConfig file, PWA allows multiple users to collaborate on the configuration of tests specifications, host groups, and configs. Users can be designated super-admins or normal users, depending on how much access they need. It is also possible to allow users to edit some configuration files, but not others.

The architecture shown in the figure 2, assumes the names of the instances as pwa-admin1, pwa-pub1, nginx mongodb, sca-auth and postfix. The user can modify and add more publishers (pwa-pub), to improve publisher performance, if needed.

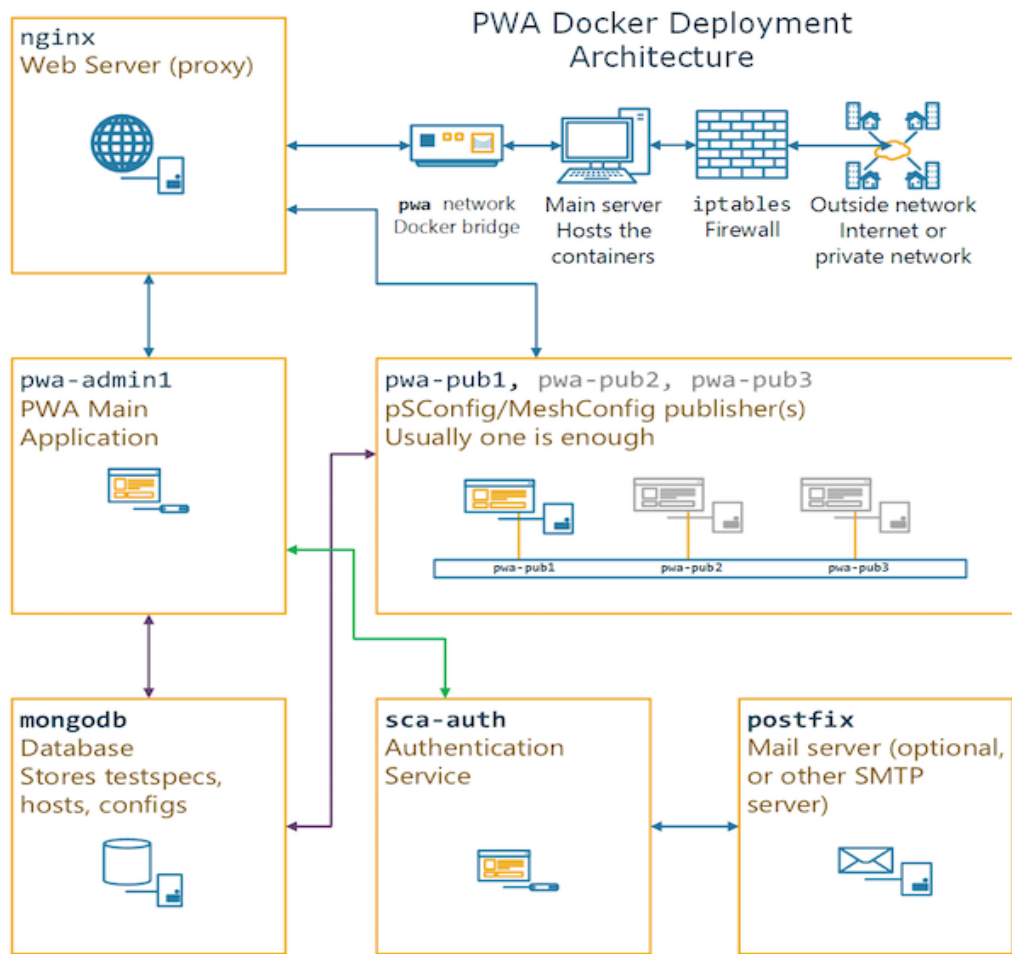


Figure 2. PWA architecture¹.

PWA is deployed using a series of docker containers some are PWA-specific and provided by the perfSONAR project. In this lab the user will use PWA interface to create a pSConfig file. This file groups perfSONAR nodes to run pScheduler tasks specified by the user. The output is published in order to accessible by all the nodes.

Table 2. Description the containers.

Container	Description
pwa-admin	PWA UI and API
pwa-pub	It is used for publishing Configs defined in PWA
sca-auth	Authentication module used by the GUI
nginx	Web server, used as a proxy to access the PWA and SCA components
mongodb	MongoDB, used by pwa-admin and pwa-pub
postfix	It is used to run a mail server in another docker container

1.1 PWA overview

The pSConfig Web Admin (PWA) provides the tools for managing pSConfig configuration files. In order to generate and publish a pSConfig file, the user goes through three parts:

- *Host Group*: A group of hosts that are user-selected that all can perform a certain type of test.
- *Test spec*: Test configuration for a test to run; this can include tool and test parameters, scheduling configuration, etc.
- *Config*: In the context of PWA, a Config is an actual test configuration that brings together Host Groups and Testspecs to generate a pSConfig output. The user can use this to configure meshes or other topologies.

2 Configuring hosts

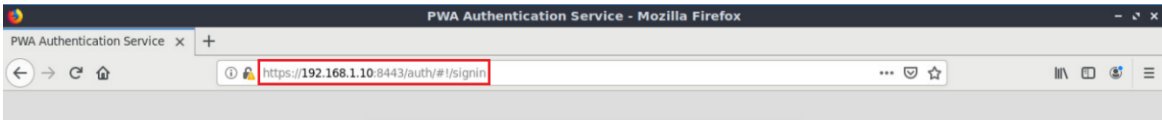
In this section, the user will configure the host information. The *Hosts* form displays a list of all perfSONAR nodes and services loaded from configured Lookup Service (sLS) data sources or defined manually (ad-hoc hosts). In this lab, the user will configure ad-hoc hosts. These hosts are perfSONAR1, perfSONAR2 and perfSONAR3. In order to proceed, the user must login the Web User Interface.

2.1 Accessing the web user interface

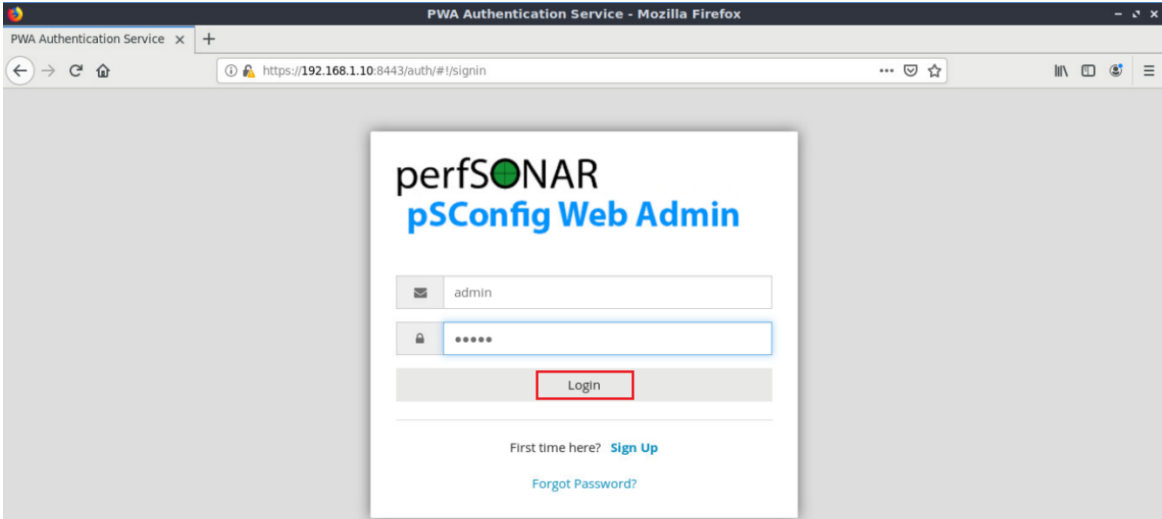
Step 1. On the Client host and open web browser.



Step 2. On the address bar, type the URL `https://192.168.1.10:8443`.

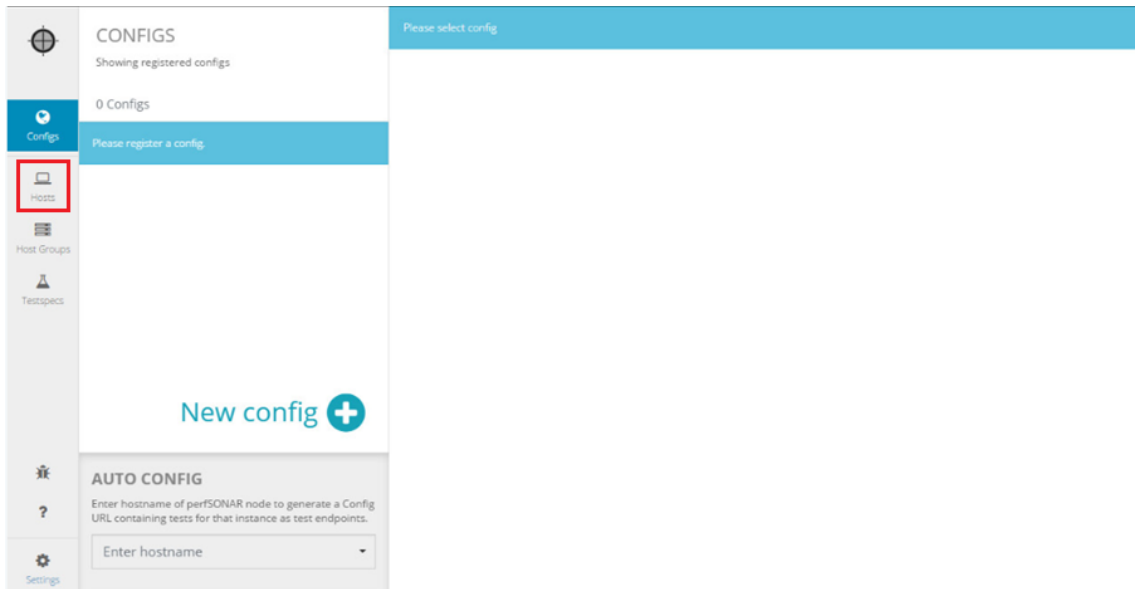


Step 3. The user will be given an authentication screen. Type `admin` as the *Username* and `admin` as the *Password*. Click on *Login*.

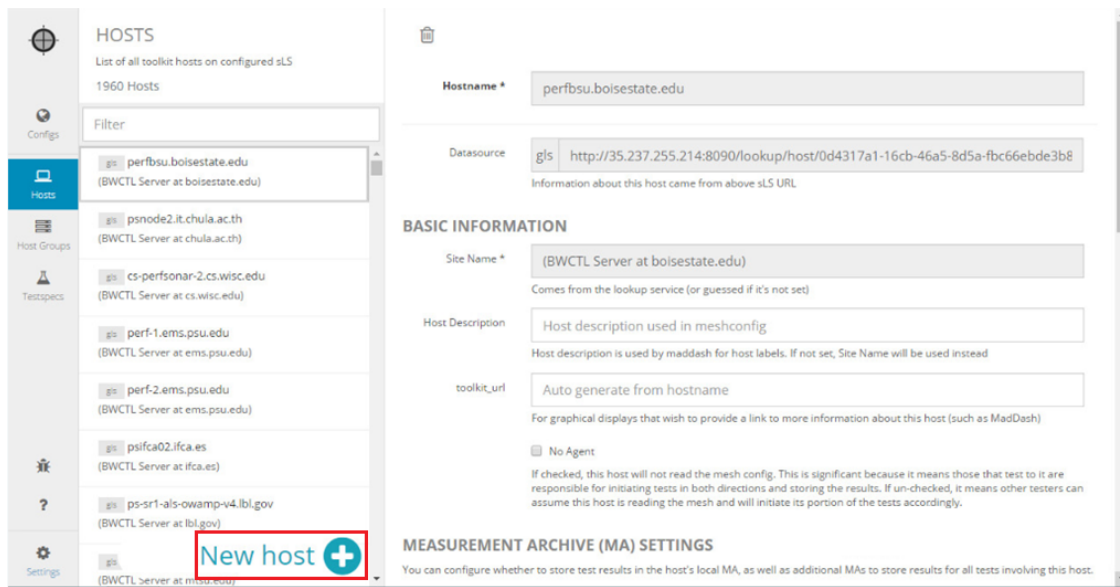


2.2 Adding hosts to the directory

Step 1. On the left part of the web interface, click on *Hosts*.



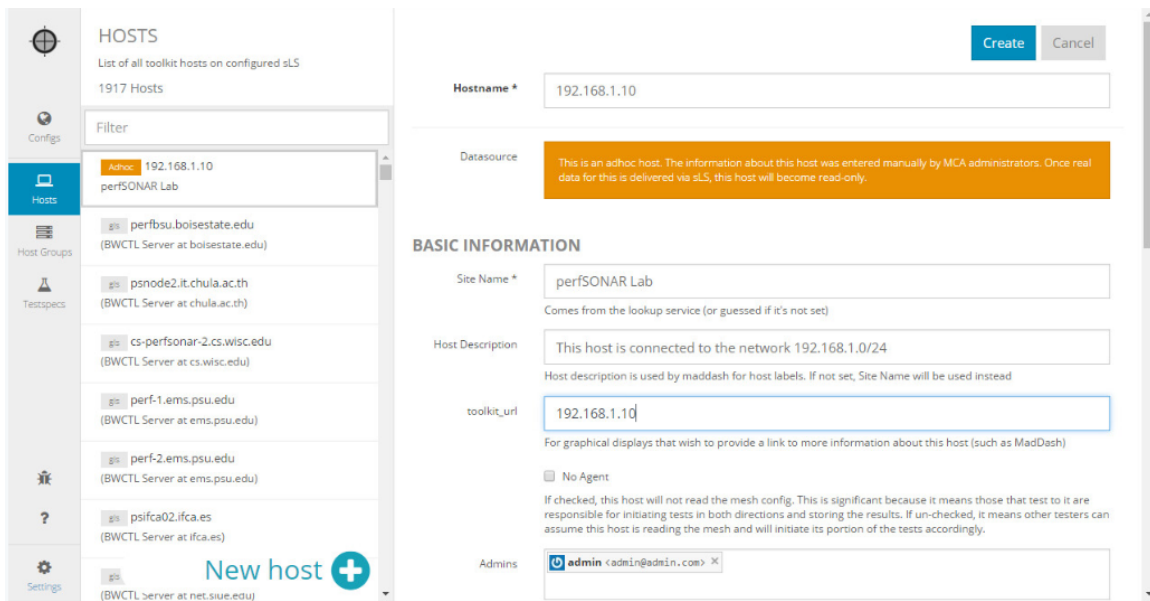
Step 2. A form will be displayed. On left side, it is displayed the list with all the public perfSONAR nodes. On the right side, it is shown all the information about the selected node. In this lab, the user will define the configuration of each host. To proceed, click on *New host*.



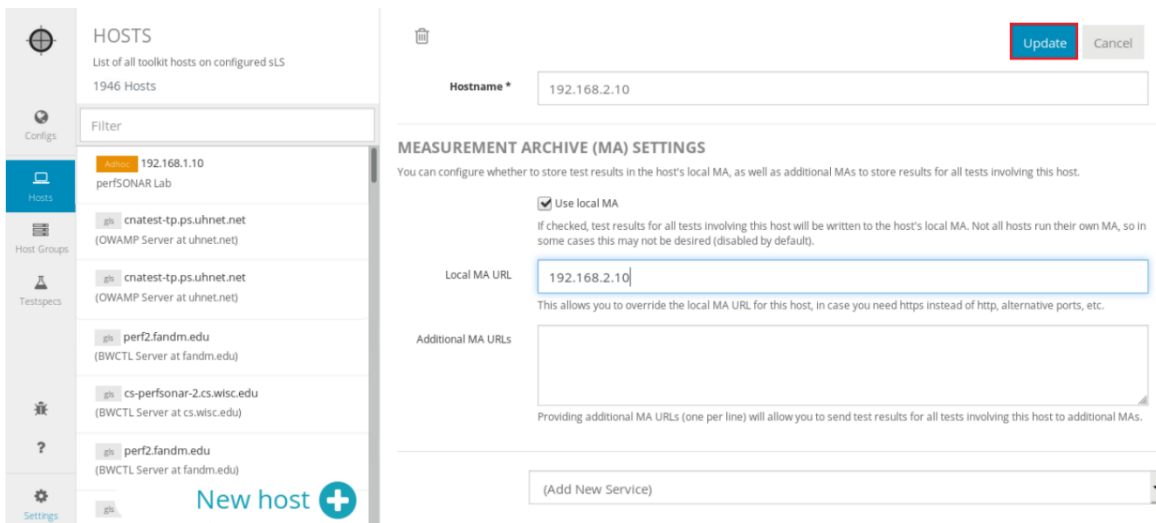
Step 3. A form will be shown up on the right side. The fields must be completed with the following information:

- **Hostname:** This label is used to identify the host on the Global Lookup Service (GLS). For this lab, complete this field with the IP addresses of perfSONAR1 (192.168.1.10), perfSONAR2 (192.168.2.10) and perfSONAR3 (192.168.3.10)
- **Site Name:** The name of the site, typically this comes from the GLS. Complete this information typing perfSONAR Lab.
- **Host Description:** This information will be displayed in MaDDash as the row/column labels. Add a brief description about the host.
- **toolkit_url:** This is the URL that links back to the toolkit instance on MaDDash matrix view. Complete this entry with the IP addresses of perfSONAR1, perfSONAR2 and perfSONAR3 respectively.

In the figure below, it is given the configuration of perfSONAR1 node (192.168.1.10). Complete the form with the information shown below, then click on *Create* to save the configuration.



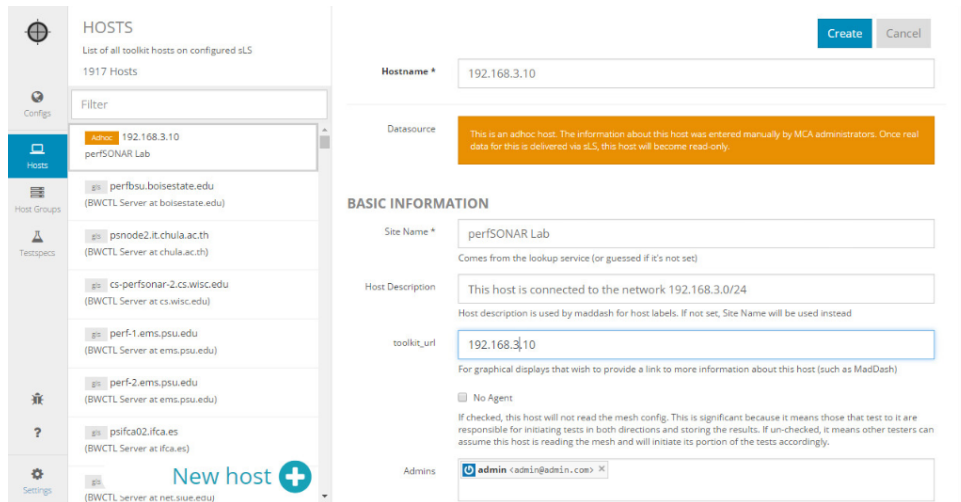
Step 4. Scroll down to add information about the measurement archive (MA). In this lab, perfSONAR2 node is configured to store the measurement data collected by each node. Check the box *Use local MA* and complete the field *Local MA URL* with the IP address of perfSONAR2 (192.168.2.10), then click on *Update* to save the configuration.



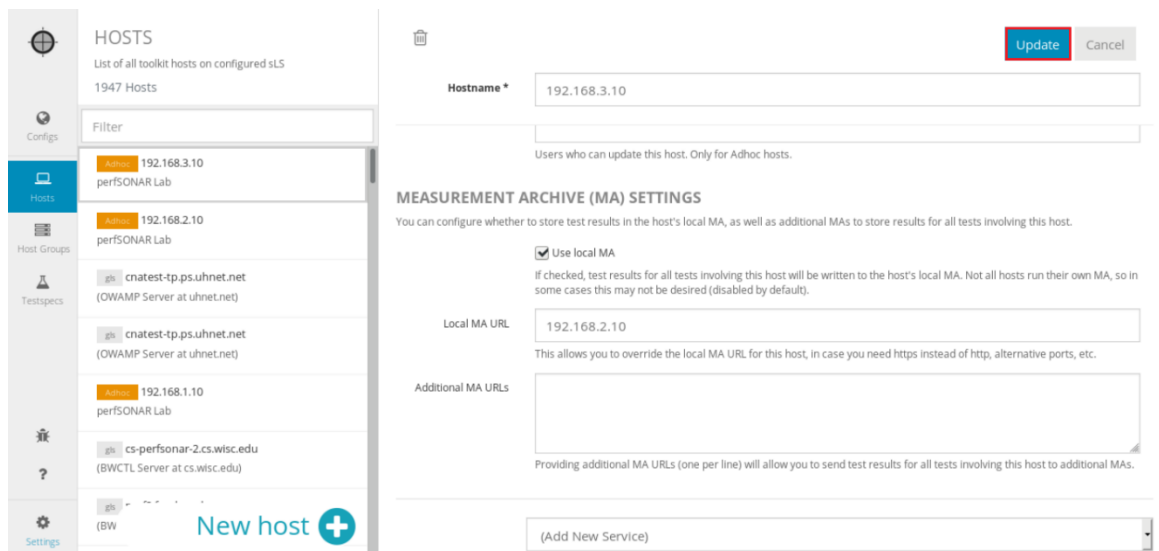
Step 5. Click again on *New host* to add information about perfSONAR2 node (192.168.2.10). Complete the form with the information shown below and click on *Create* to save the configuration.

Step 6. Scroll down to add information about the measurement archive (MA). In this lab, perfSONAR2 node is configured to store the measurement data collected by each node. Check the box *Use local MA*, then complete the field *Local MA URL* with the IP address of perfSONAR2 (192.168.2.10), then click on *Update* to save the configuration.

Step 7. Click again on *New host* to add information about perfSONAR3 node (192.168.3.10). Complete the form with the information shown below and click on *Create* to save the configuration.



Step 8. Scroll down to add information about the measurement archive (MA). In this lab, perfSONAR2 node is configured to store the measurement data collected by each node. Check the box *Use local MA*, then complete the field *Local MA URL* with the IP address of perfSONAR2 (192.168.2.10), then click on *Update* to save the configuration.

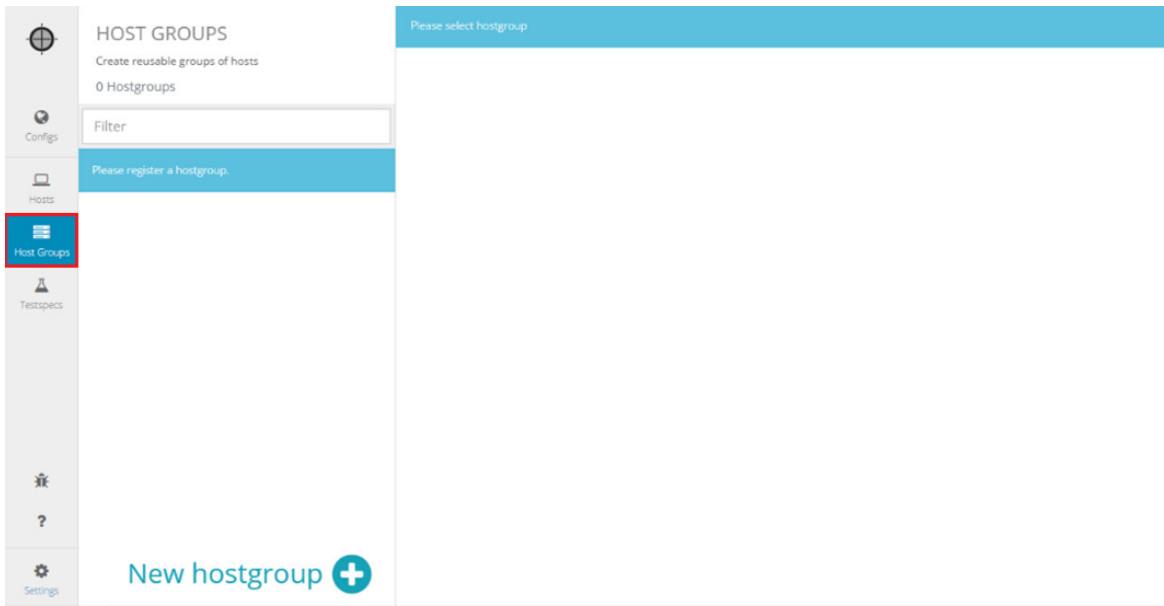


3 Configuring host groups

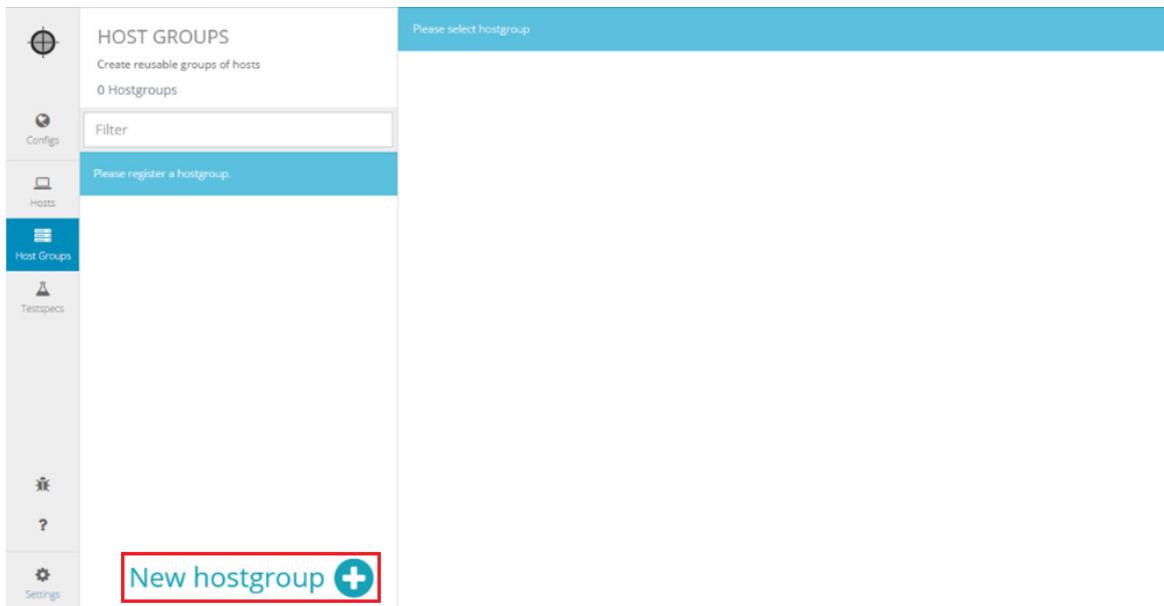
A host group is a logical grouping of perfSONAR nodes. The user may reuse a single host group for multiple configuration files. In this section the user will configure two groups of perfSONAR nodes. Both groups include perfSONAR1, perfSONAR2 and perfSONAR3. The first group is for throughput tests, and the second group is for latency measurements.

3.1 Configuring Throughput Group

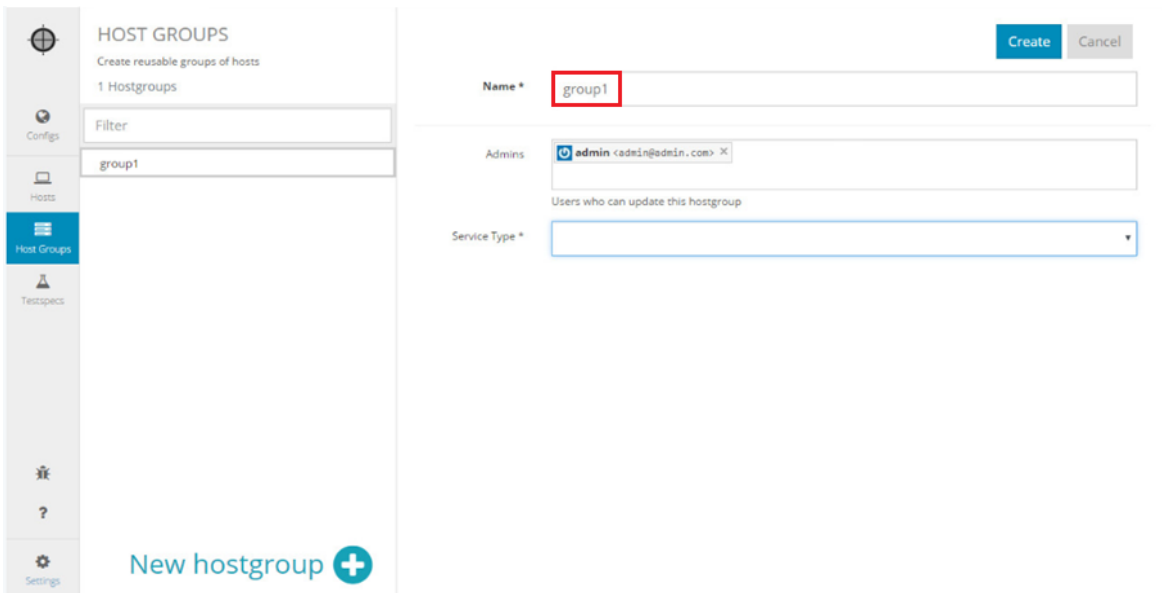
Step 1. On the left part of the web interface, click on *Host Groups*.



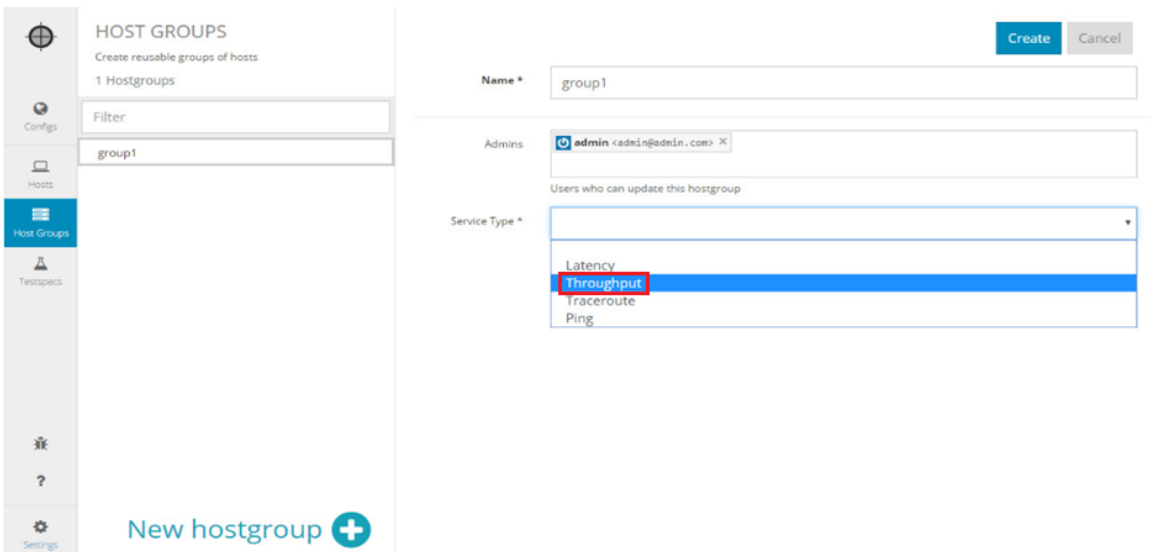
Step 2. Click on *New hostgroup*.



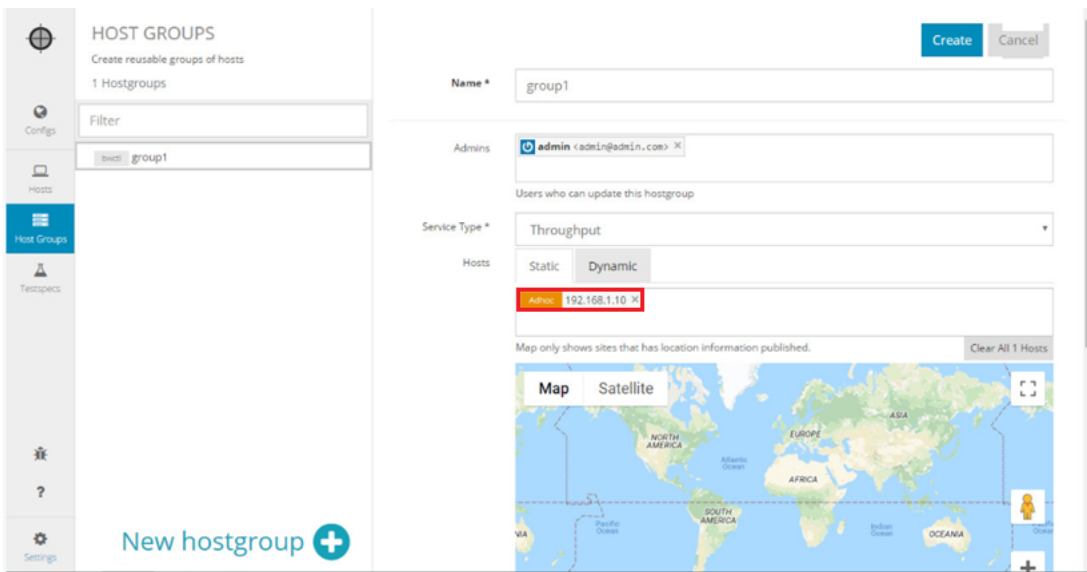
Step 3. Write *group1* as the name. This name will be a tag to identify the host group during the test configuration.



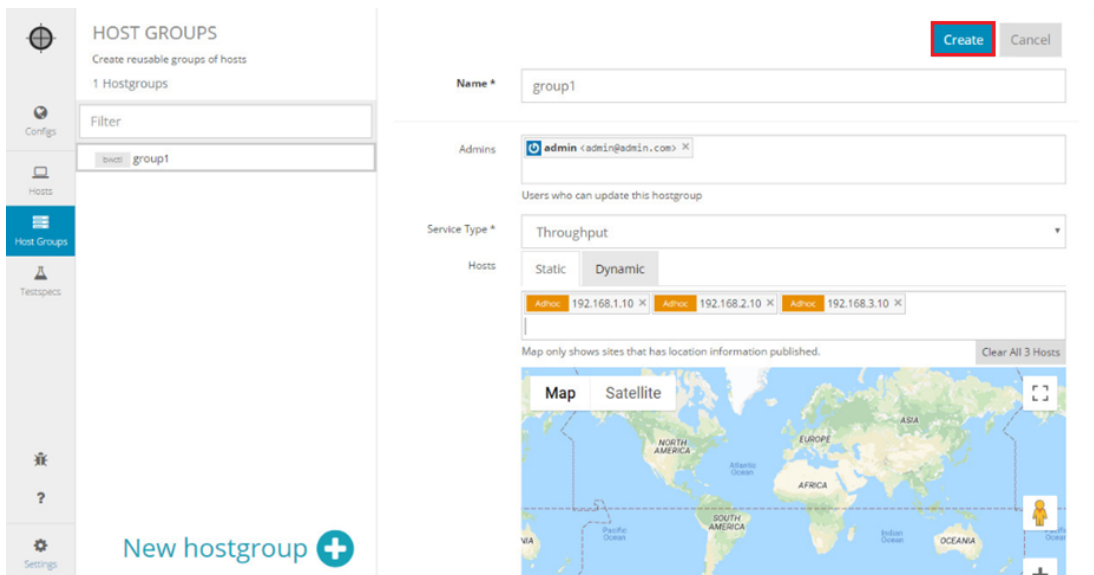
Step 4. Click on *Service Type*. A list will be displayed, select *Throughput*.



Step 5. On *Hosts* field, type the IP address of perfSONAR1 node (192.168.1.10) to search for the host configured on the last section. A list will be displayed, select 192.168.1.10 to add the node to the group.

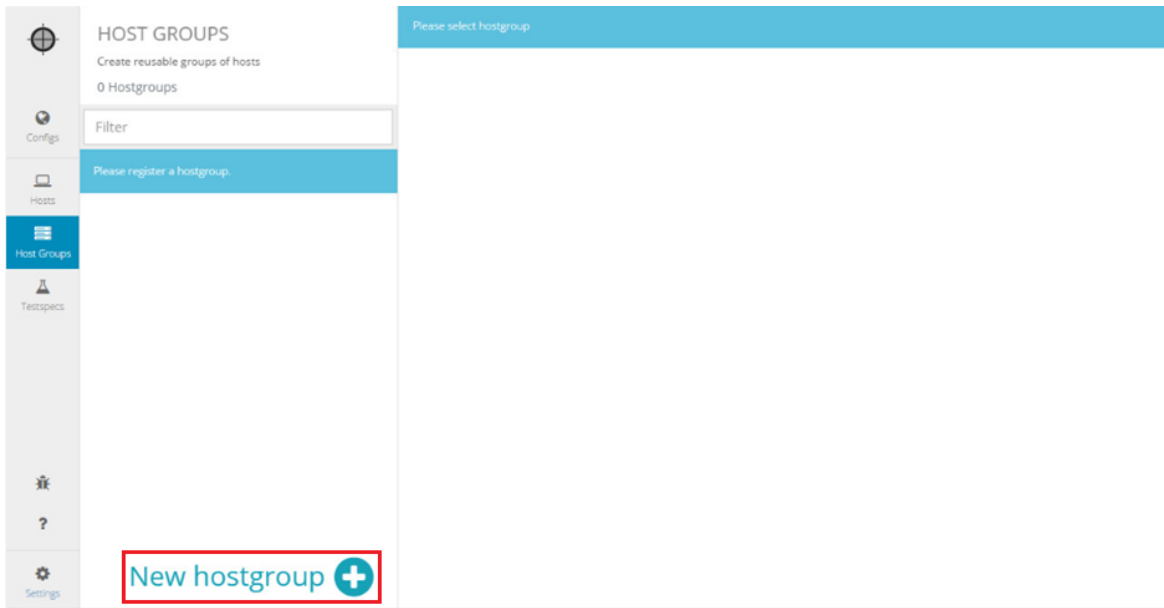


Step 6. Repeat the previous step but now, complete the form with perfSONAR2 (192.168.2.10) and perfSONAR3 (192.168.3.10) IP addresses. Then, click on *Create* to save the configuration.

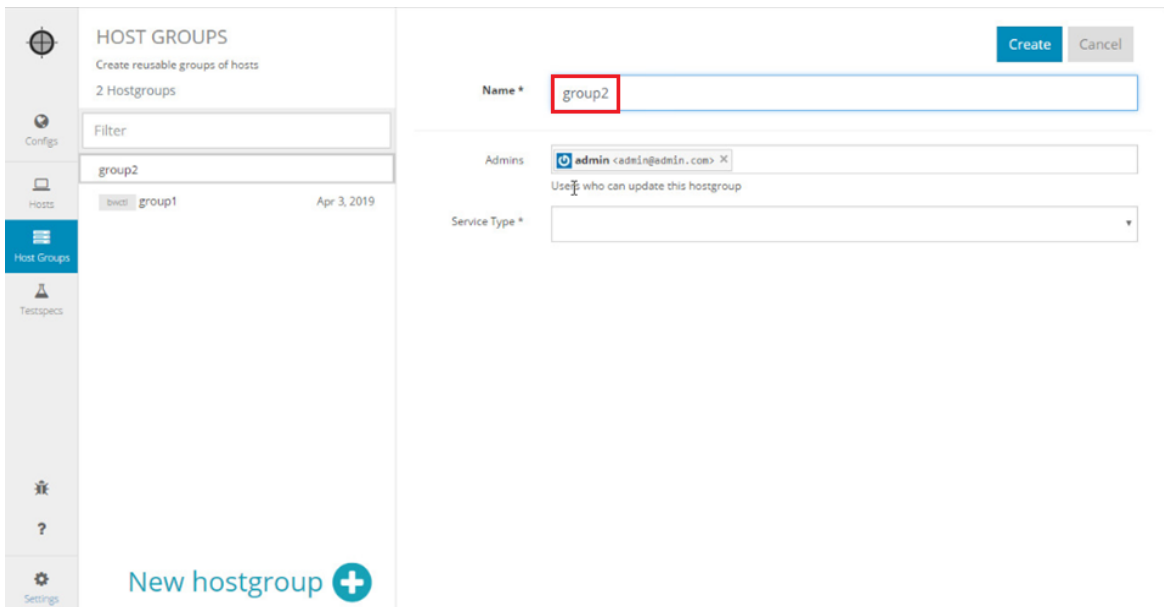


3.2 Configuring latency group

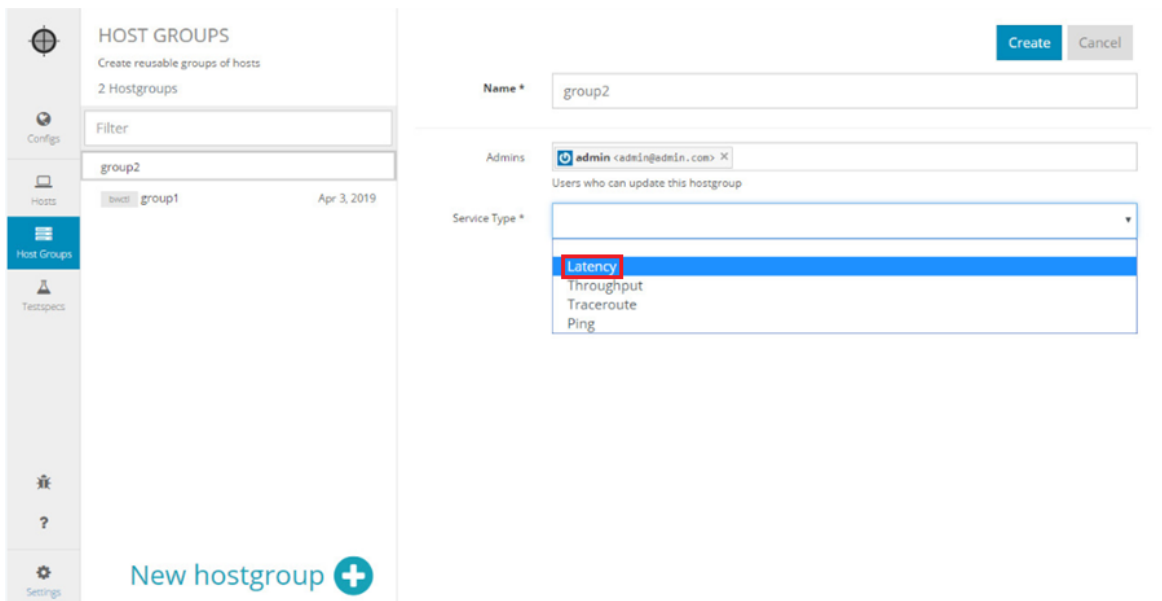
Step 1. In order to create a new group, click again on *New hostgroup*.



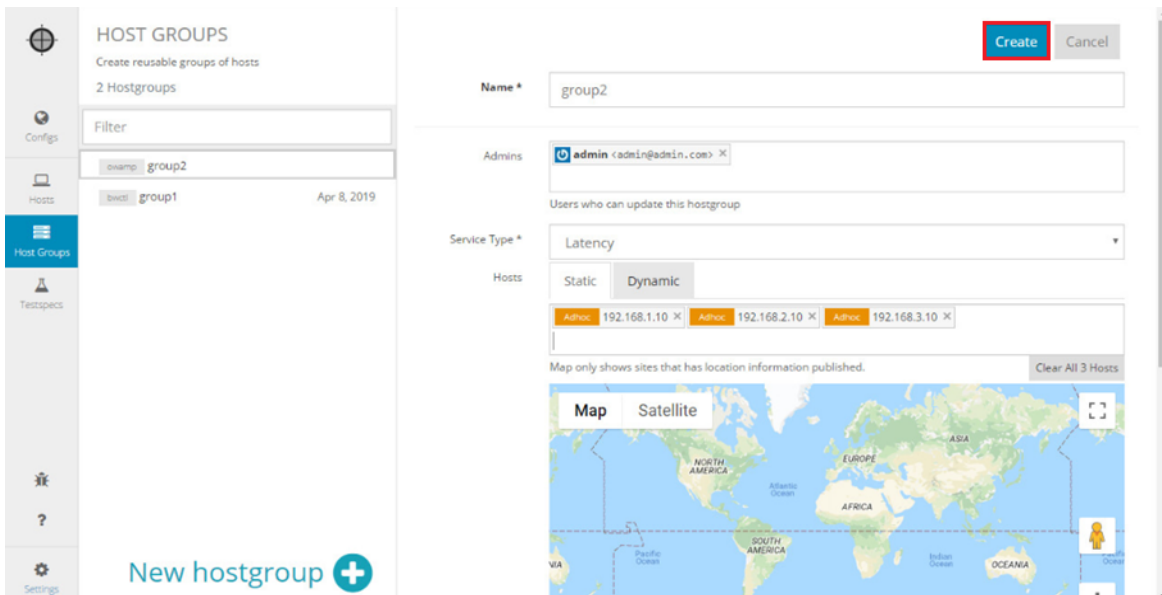
Step 2. Write *group2* as the name. This will be a tag to identify the host group during the test configuration



Step 3. Click on *Service Type*. A list will be displayed, select *Latency*.



Step 4 As in the previous section, on the *Hosts* field, type the IP address of perfSONAR1 node (*192.168.1.10*) to search for the host configured on the last section. A list will be displayed, select *192.168.1.10* to add the node to the group. In addition, add perfSONAR2 (*192.168.2.10*) and perfSONAR3 (*192.168.3.10*) IP addresses. Then, click on *Create* to save the configuration.

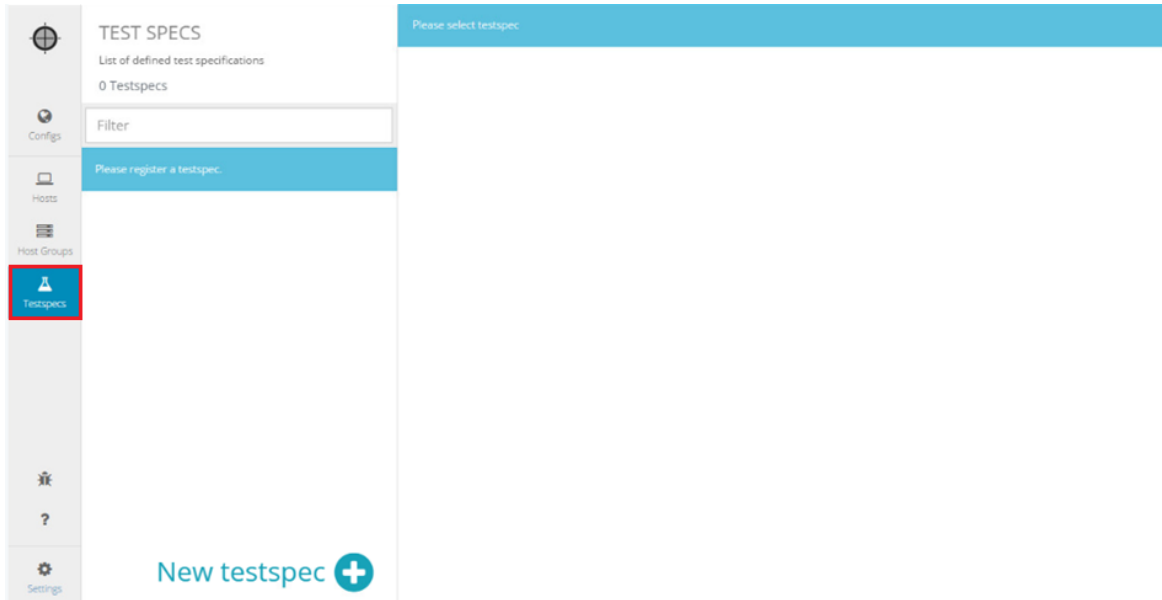


4 Setting test specifications

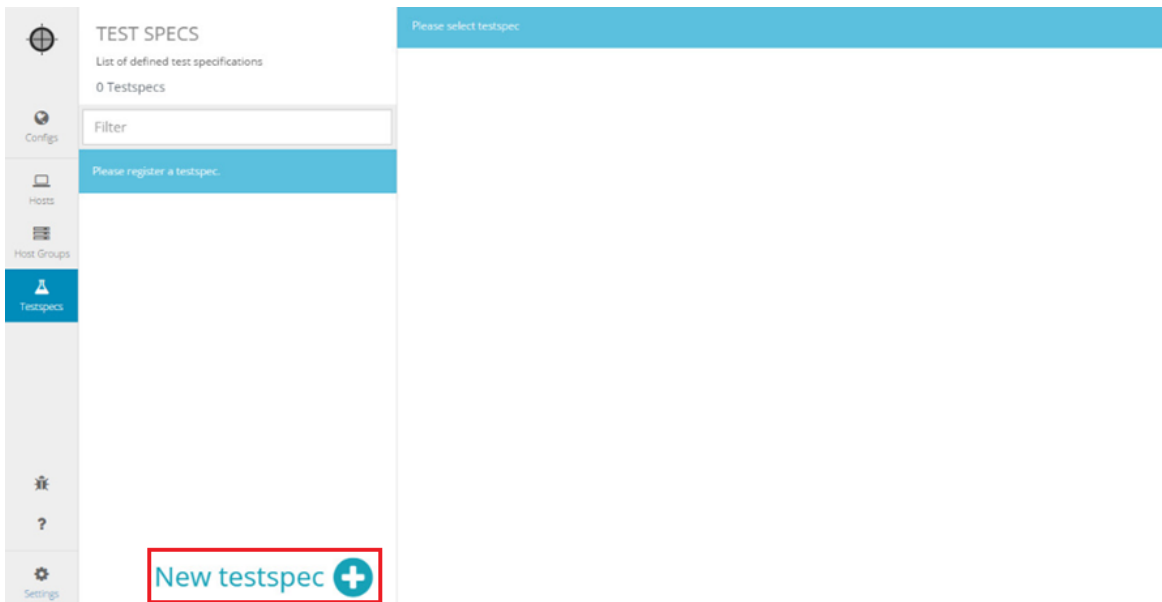
The test specification is a set of parameters used by a particular test service. Instead of defining such parameters for each test, the user can define and use them in one or more configuration definitions. In this section the user will configure the tests specification for the host groups created on the last section. The first test specification is for throughput test which corresponds to the group1. The second test specification is for the latency test which corresponds to the *group2*.

4.1 Configuring throughput test specification

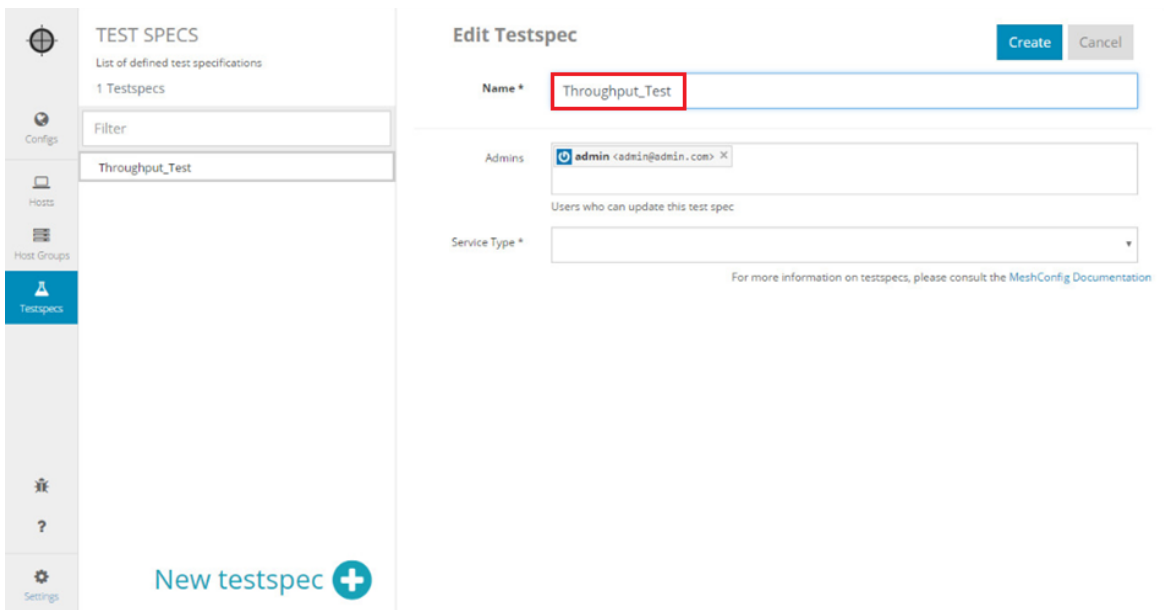
Step 1. On the left part of the web browser, click on *Testspec*.



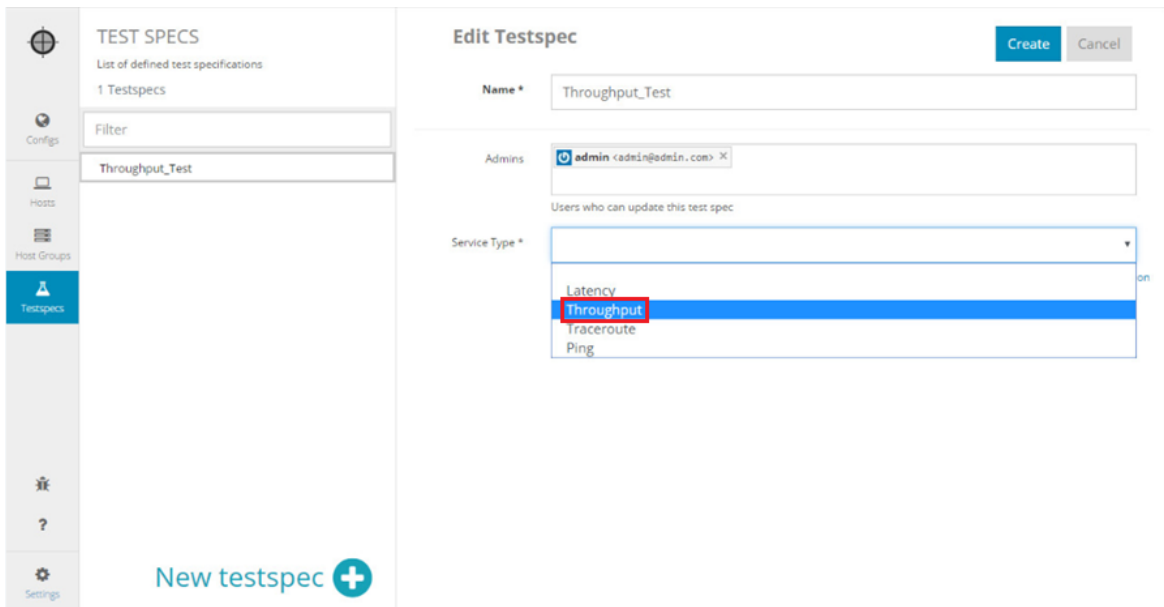
Step 2. Click on *New testspec*.



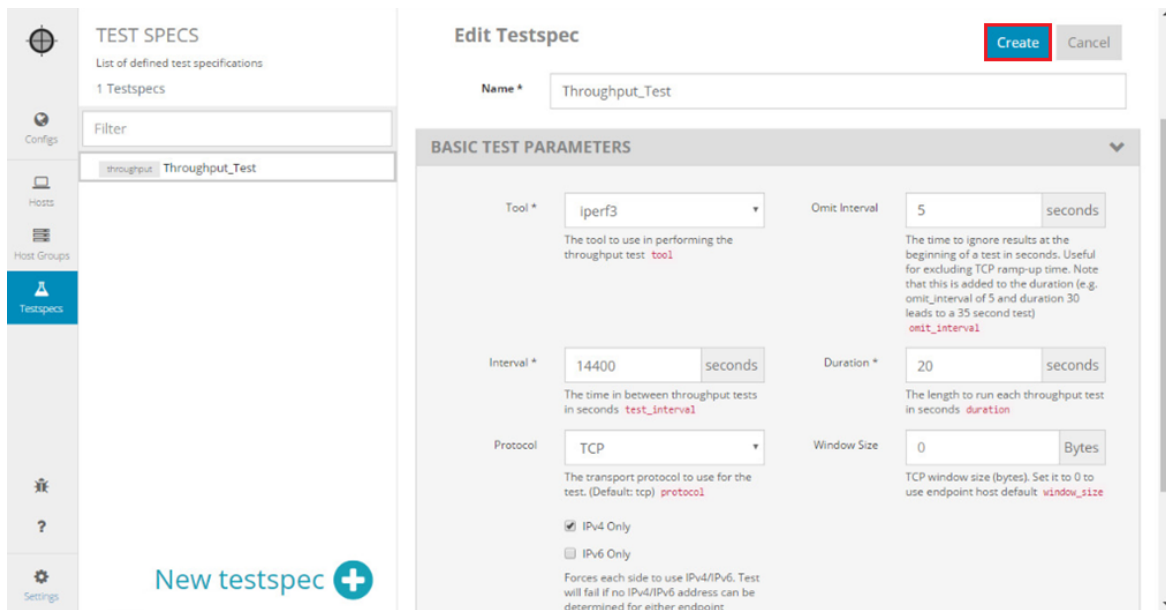
Step 3. Type *Throughput_Test* in the *Name* field.



Step 4. Click on *Service Type*, a list will be displayed. Select *Throughput* in order to configure a throughput test.

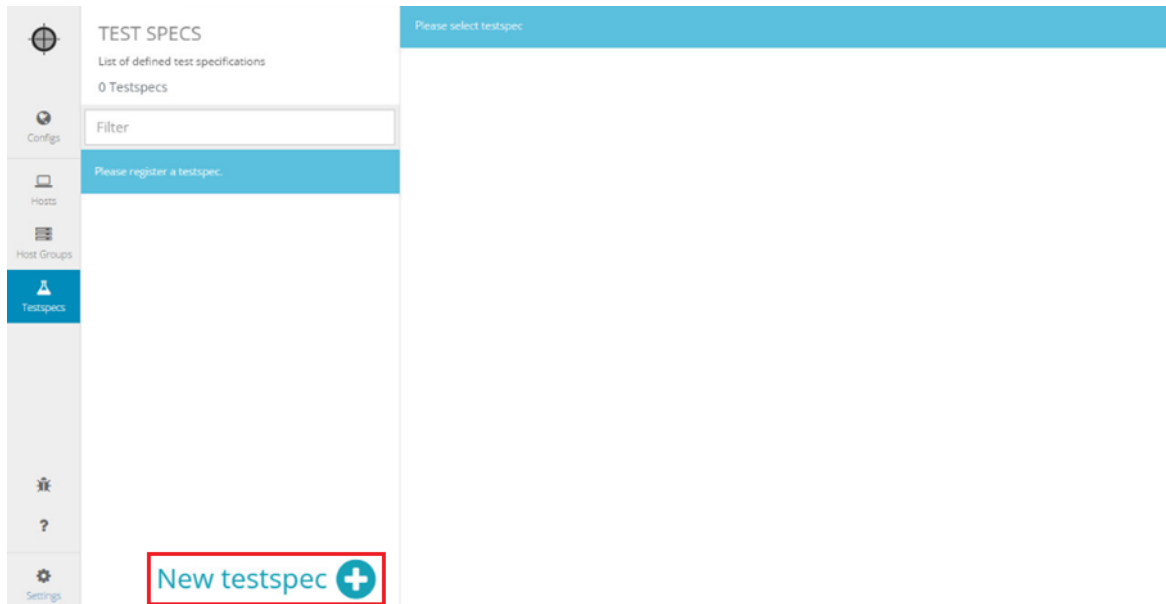


Step 5. A parameter box will be displayed after selecting the *Service Type*. The user will see different set of test parameters. Below each parameter is shown the description of each field. In this lab, the user will use the default configuration. Click on *Create* to save the configuration.

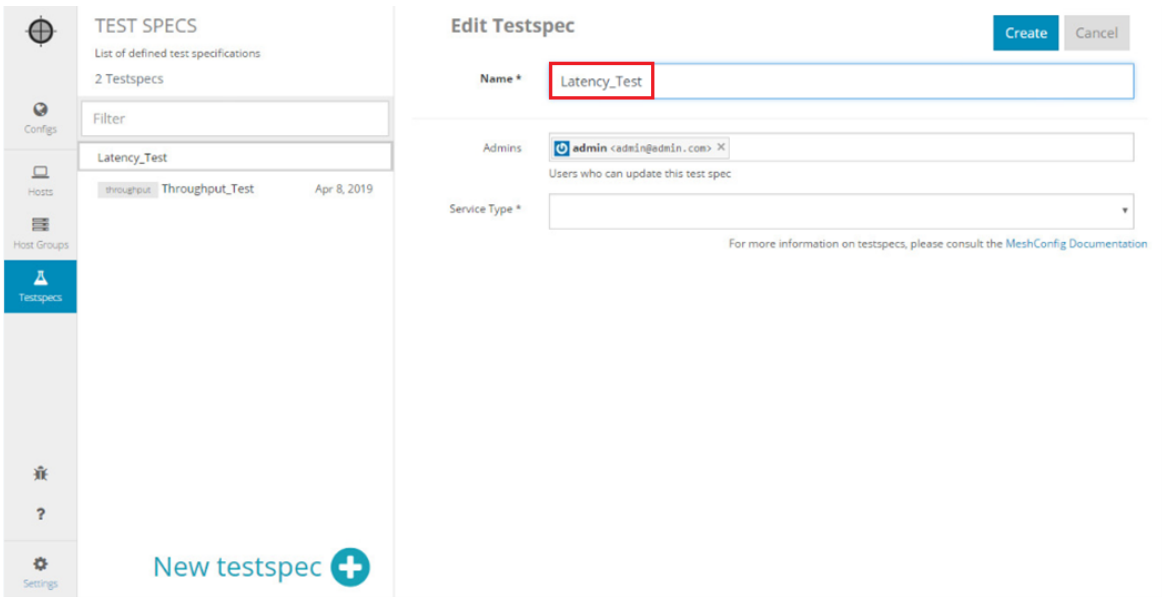


4.2 Configuring latency test specification

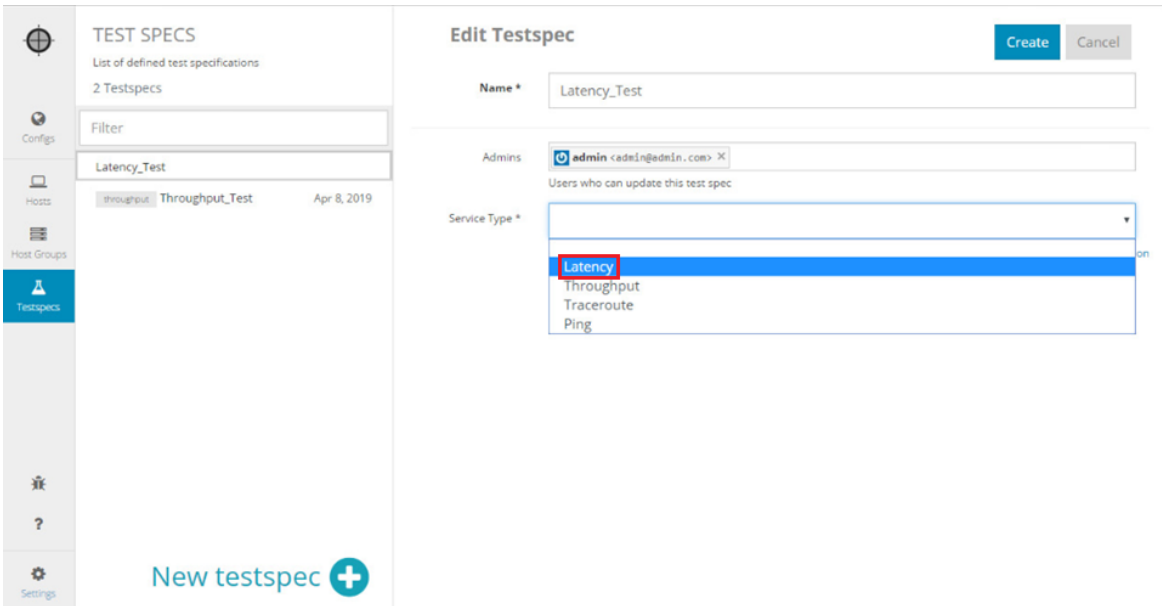
Step 1. Click on *New Testspec.*



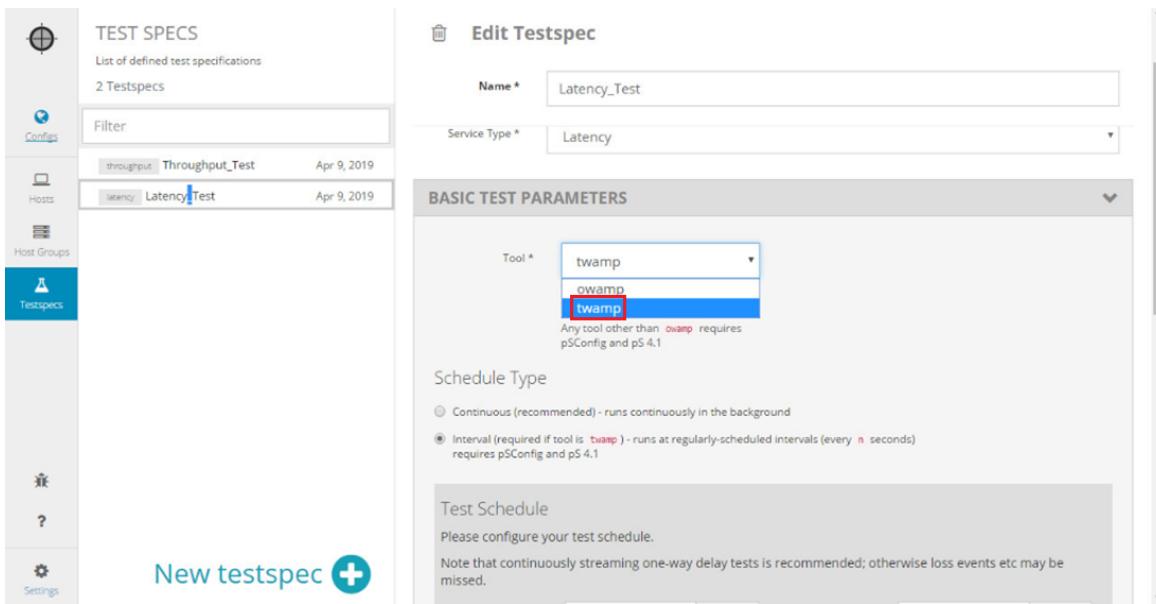
Step 2. Type *Latency_Test* in the *Name* field.



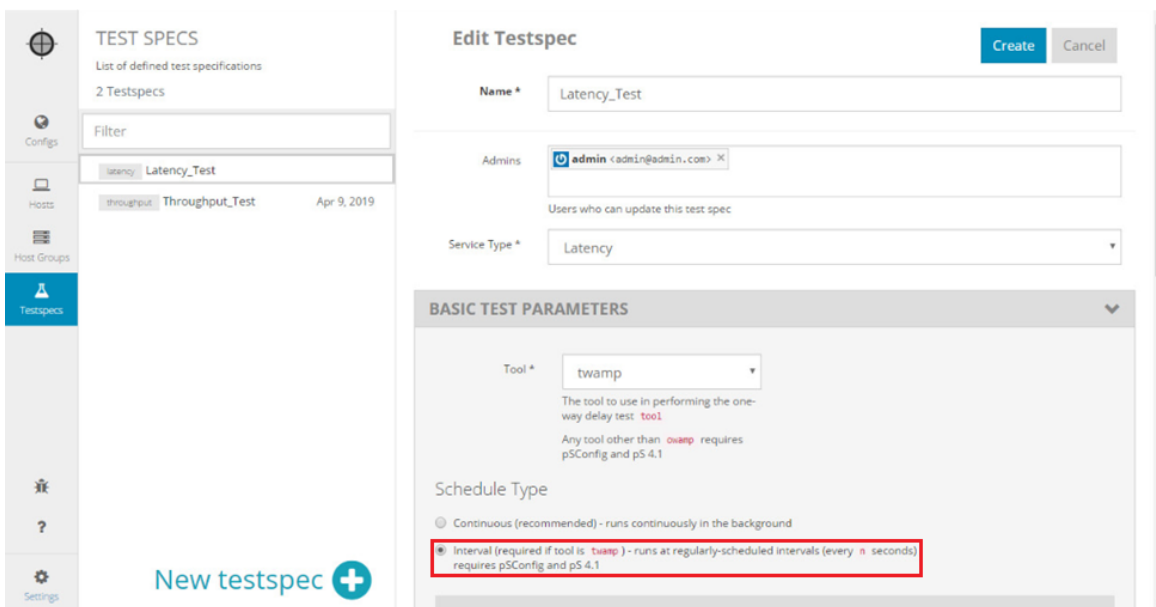
Step 3. Click on *Service Type*, a list will be displayed. Select *Latency* in order to configure a Latency test.



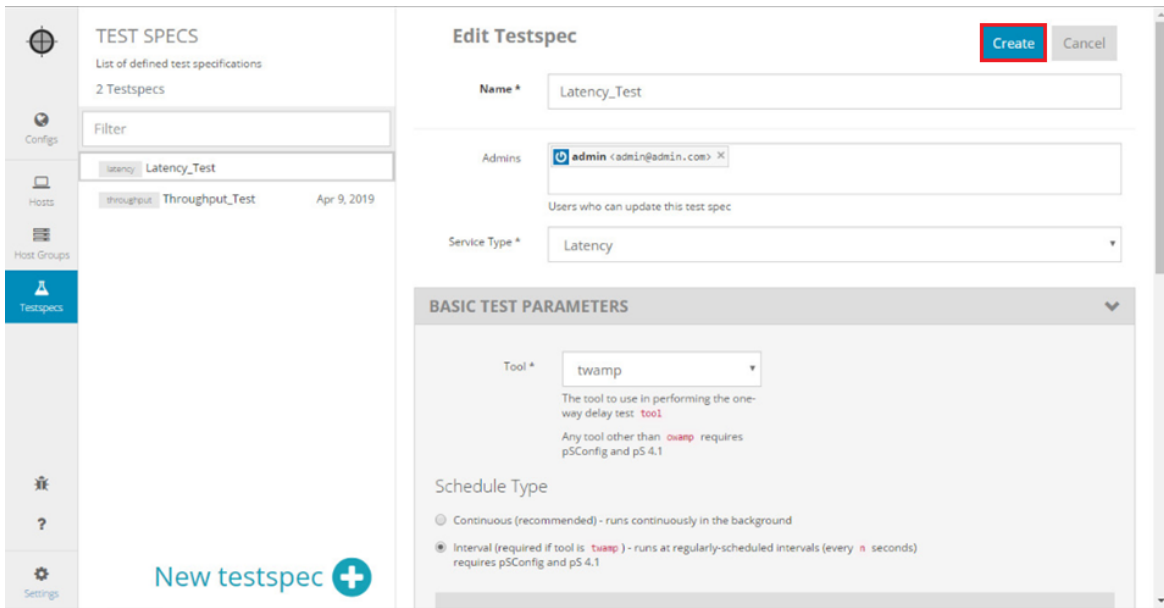
Step 4. A parameter box will be displayed after selecting the *Service Type*. The user will see different set of test parameters. Below each parameter is shown the description of each field. Select *twamp* as the *Tool*.



Step 5. On the *Schedule Type*, select the second option as shown in the figure below.



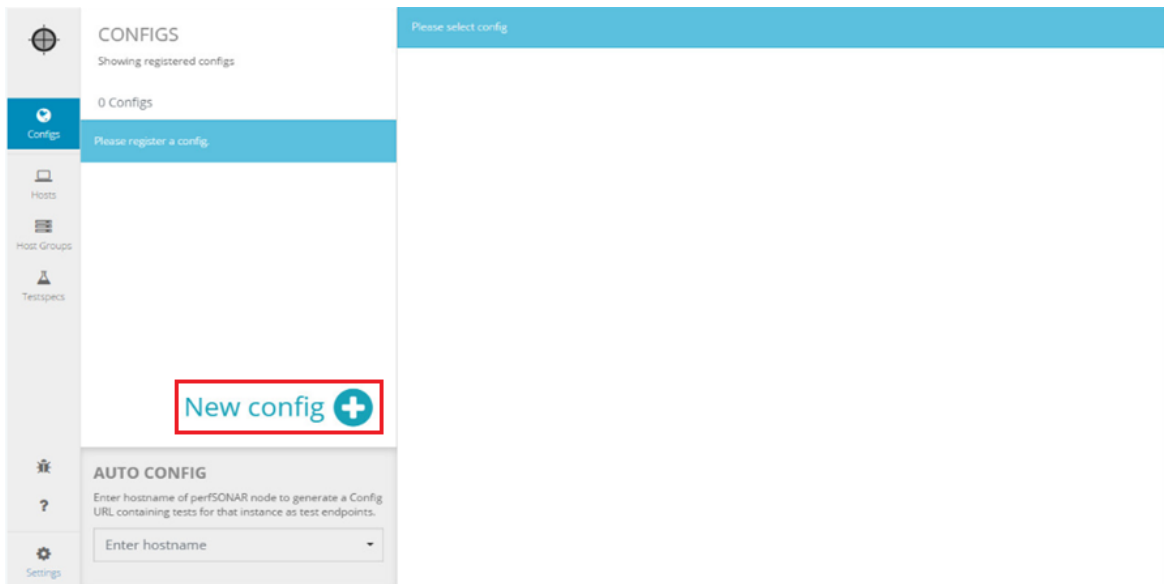
Step 6. Click on *Create* to save the Configuration.



5 Creating pSConfig output

Once the Host Groups and Test Specs are defined, it is possible now to create pSConfig file by combining those entities. Under the *Config* section, the user will see a list of Configs defined and their basic information. The links displayed next to the *Config* name is the actual Config URLs that users can download and subscribe on various perfSONAR services. To edit, or see more detail for each Config, click on the *Config* name in the Configs column.

Step 1. In order to define a configuration file, click on *New config*.



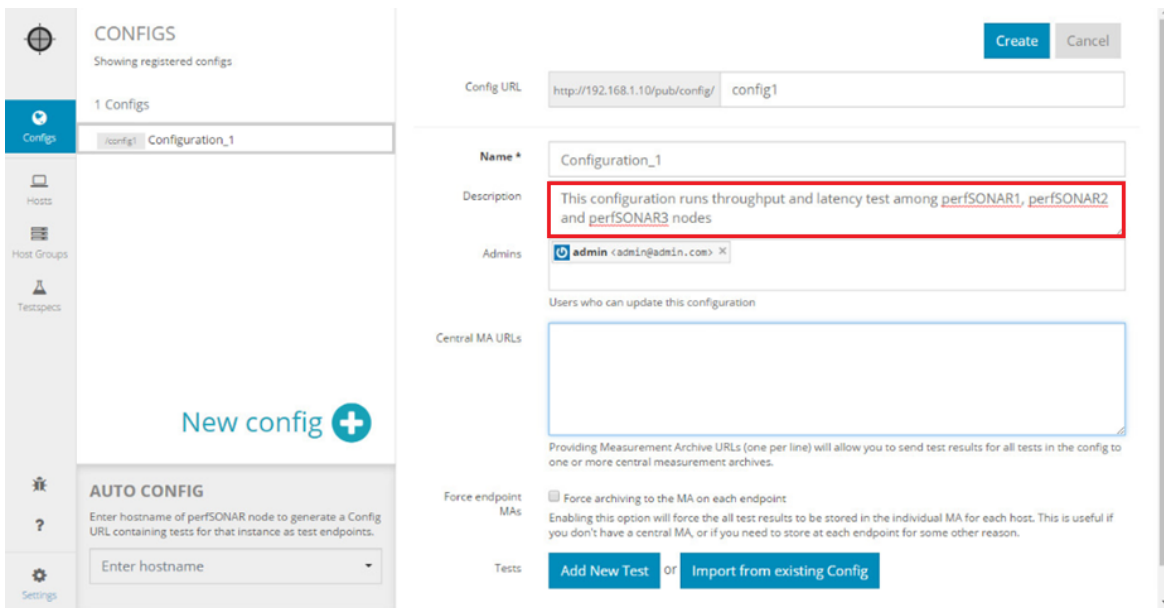
Step 2. The *Config URL* shows the url where the configuration file will be published once the configuration is finished. Complete the entry box typing *config1*.

The screenshot shows the 'CONFIGS' section of the pSConfig Web Administrator. The left sidebar contains navigation options: Configs, Hosts, Host Groups, Testspecs, and Settings. The main content area shows a list of 1 Configs with a 'New config' button. The 'AUTO CONFIG' section is also visible. The right-hand form is for creating a new config. The 'Config URL' field is pre-filled with 'http://192.168.1.10/pub/config/' and 'config1'. The 'Name' field is empty. The 'Description' field contains 'New Config'. The 'Admins' field contains 'admin <admin@edeIn.com>'. The 'Tests' section has 'Add New Test' and 'Import from existing Config' buttons.

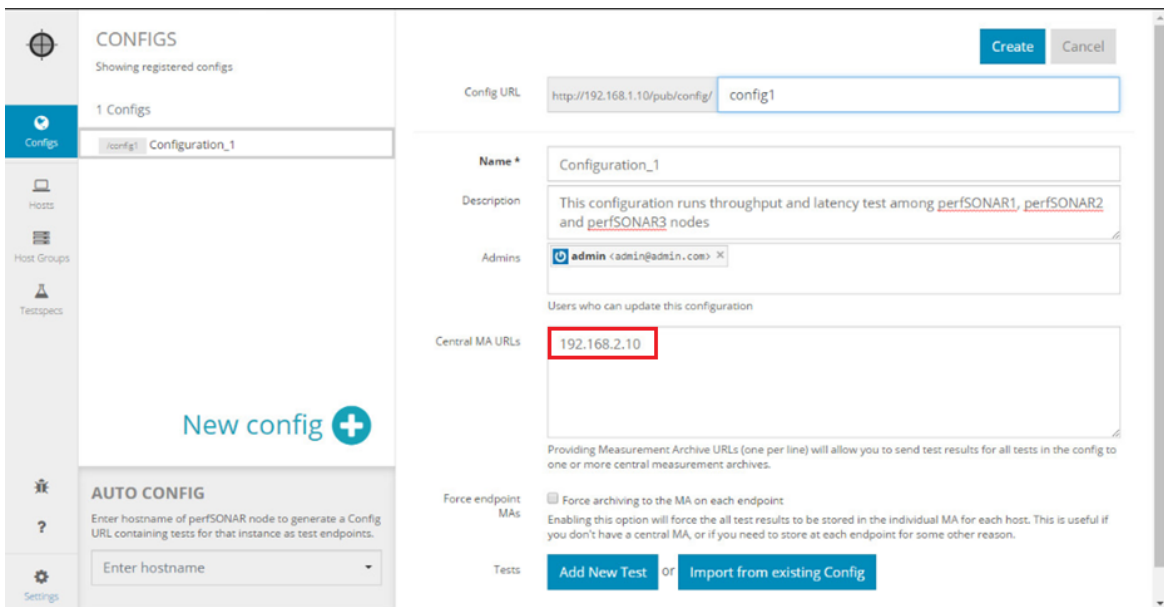
Step 3. Type *Configuration_1* as the *Name*.

The screenshot shows the 'CONFIGS' section of the pSConfig Web Administrator. The left sidebar contains navigation options: Configs, Hosts, Host Groups, Testspecs, and Settings. The main content area shows a list of 1 Configs with a 'New config' button. The 'AUTO CONFIG' section is also visible. The right-hand form is for creating a new config. The 'Config URL' field is pre-filled with 'http://192.168.1.10/pub/config/' and 'config1'. The 'Name' field contains 'Configuration_1'. The 'Description' field contains 'New Config'. The 'Admins' field contains 'admin <admin@edeIn.com>'. The 'Tests' section has 'Add New Test' and 'Import from existing Config' buttons.

Step 6. In the description box, add a brief description about the configuration file. This field is optional.

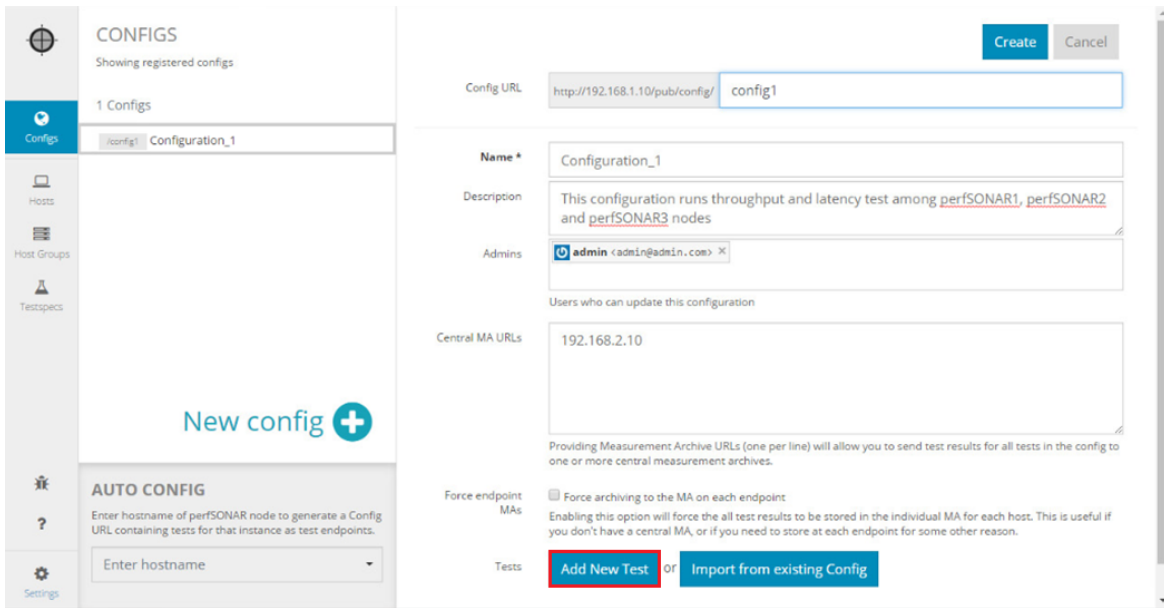


Step 7. The perfSONAR2 node is configured as the Measurement Archive (MA). This node collects the measurement data of all perfSONAR node. Type the IP address of perfSONAR2 in order to configure it as the MA.

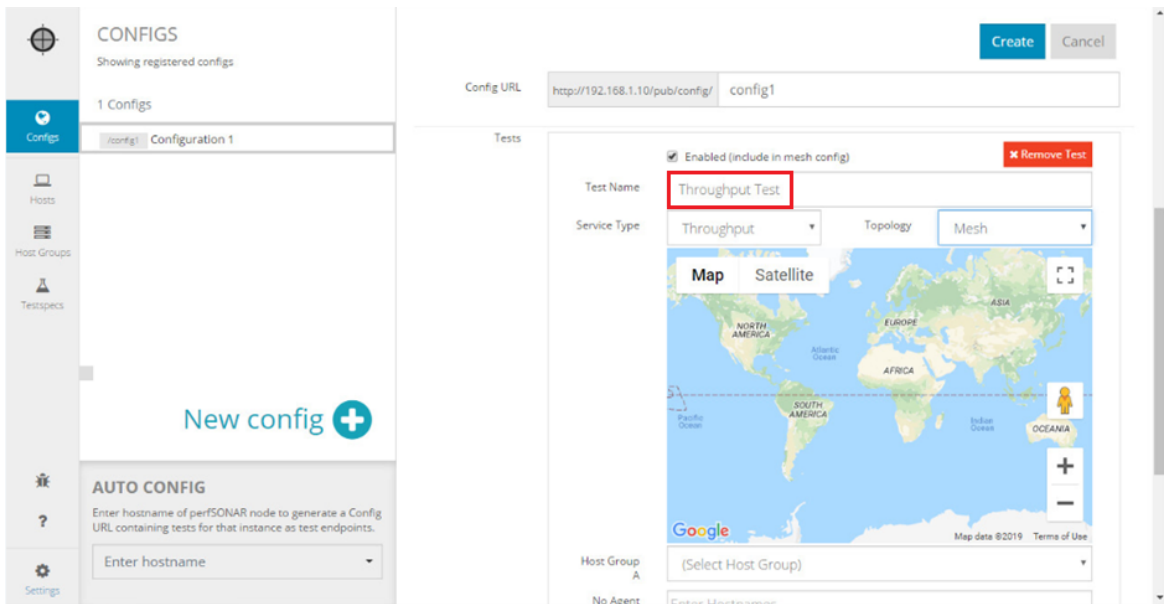


5.1 Adding throughput test

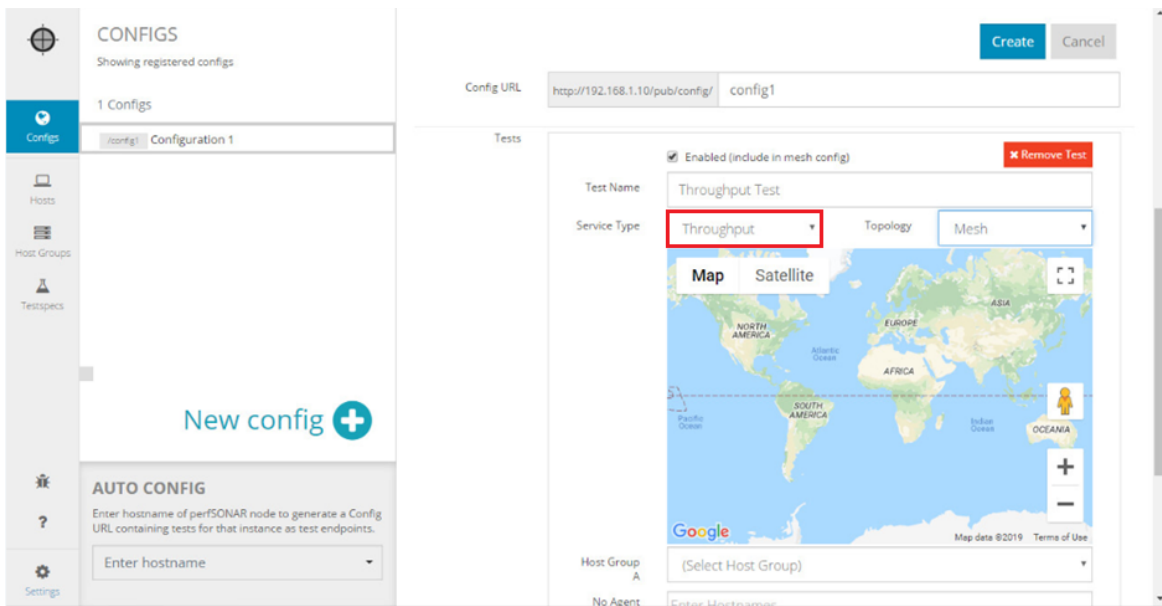
Step 1. To proceed, click on *Add New Test*. A configuration box will be displayed below.



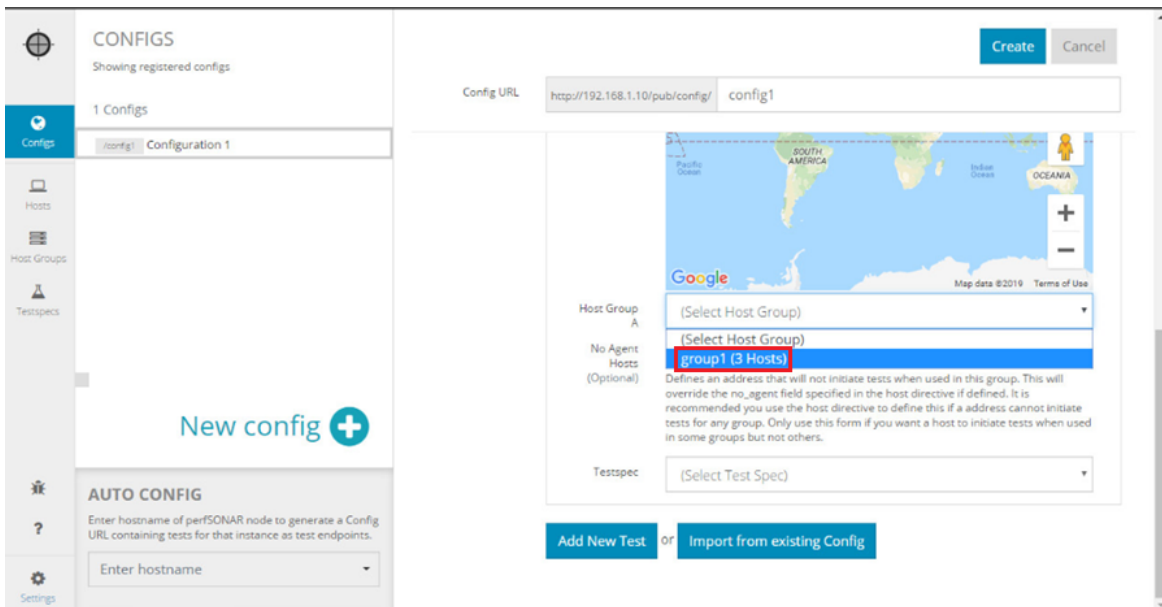
Step 2. Type *Throughput Test* as the *Test Name*.



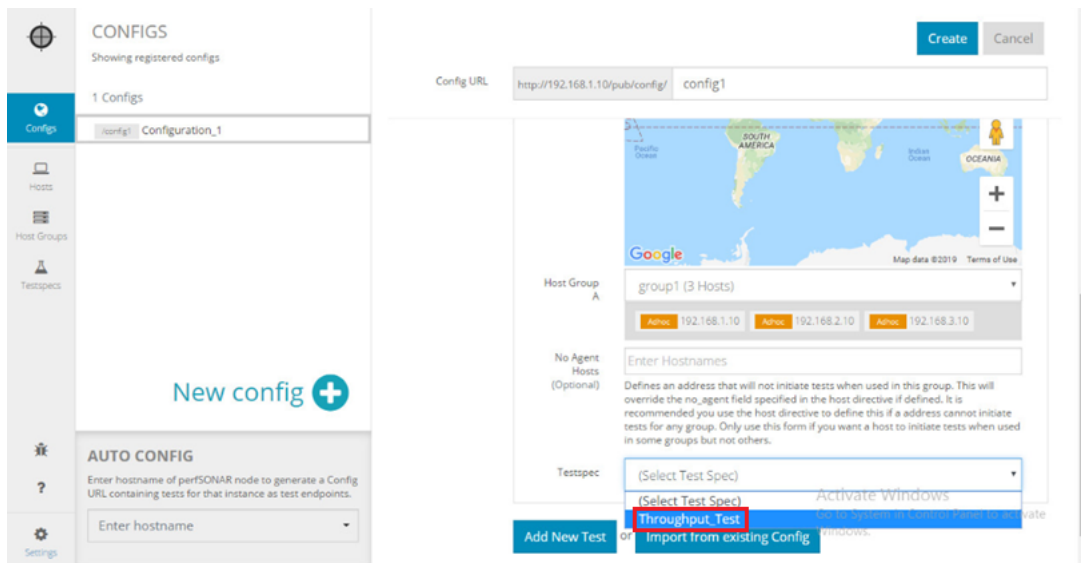
Step 3. Click on *Service Type*, a list will be displayed, select *Throughput*.



Step 4. Scroll down and click *Host Group A*, a list will be displayed, select *group1*. The user will see the IP addresses of the three nodes involved in the test.



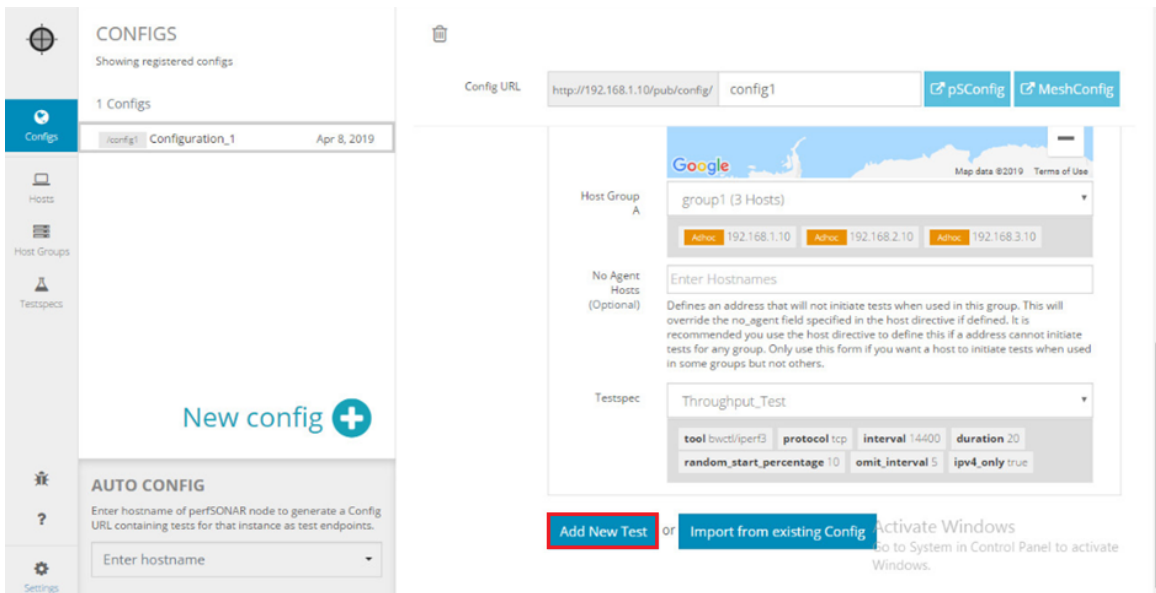
Step 5. Click on *Testspec*, a list will be displayed, select *Throughput_Test*.



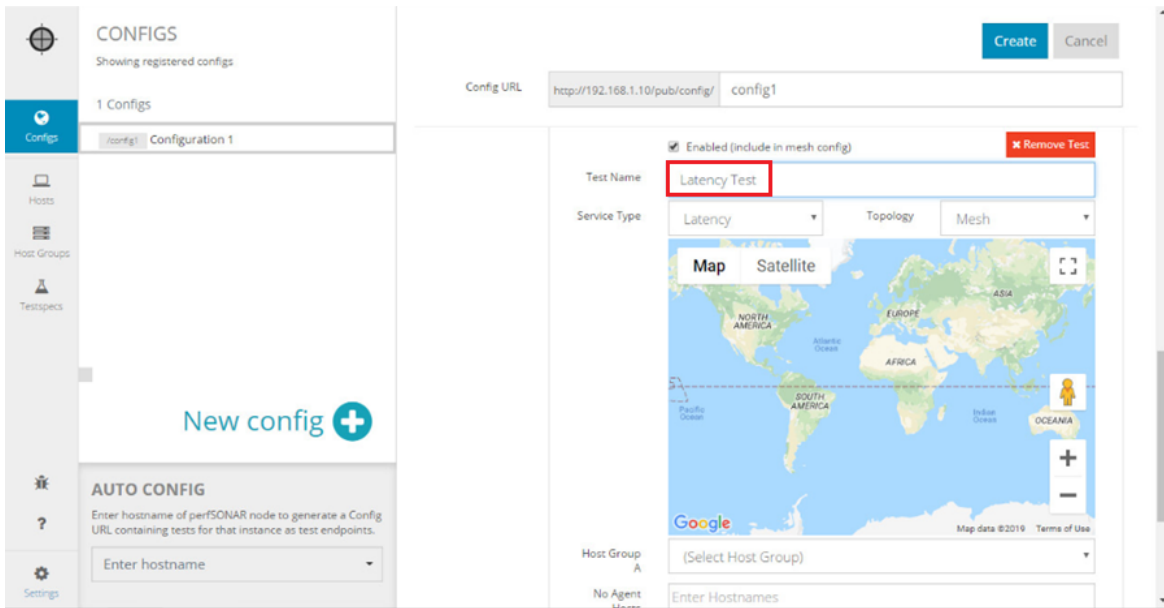
Step 6. Click on *Create* to save the changes.

5.2 Adding latency test

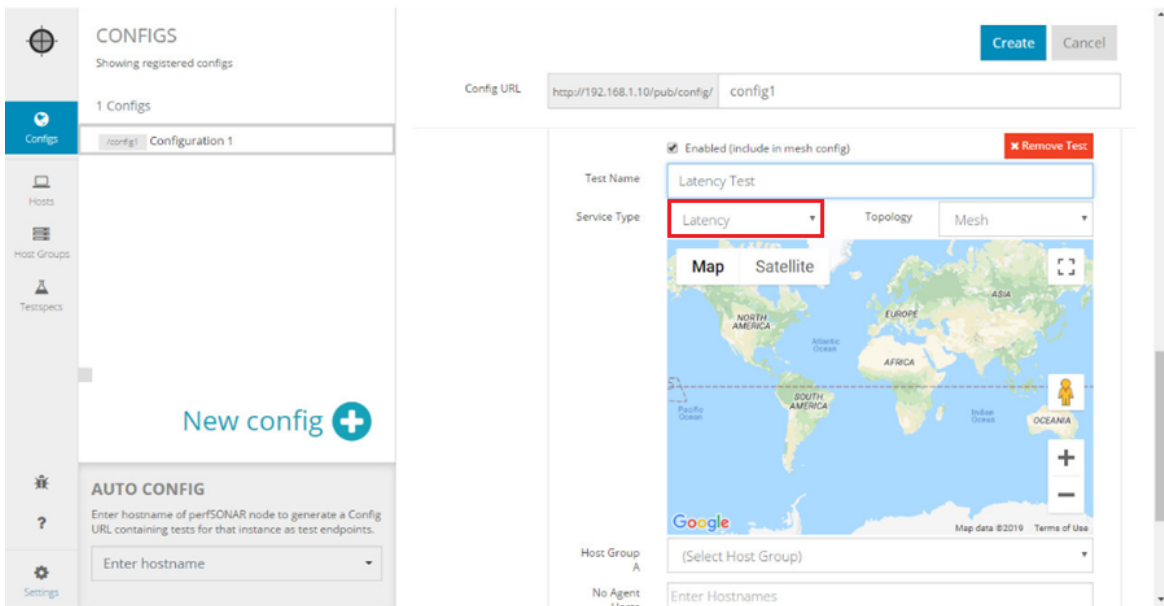
Step 1. To configure the latency test, click again on *Add New Test*.



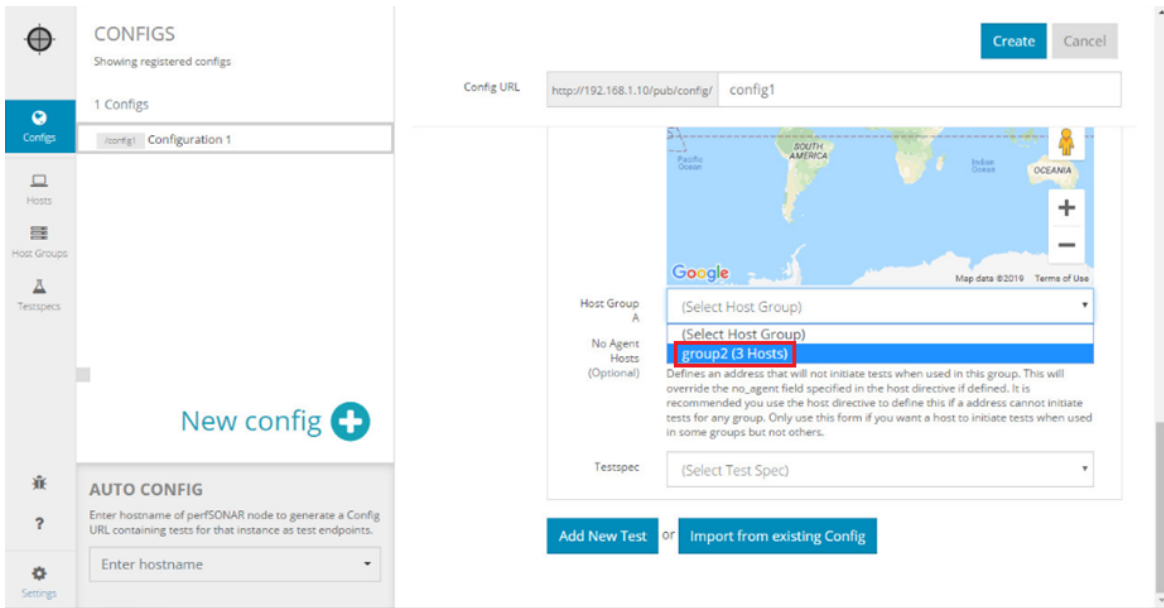
Step 2. Type *Latency Test* as the *Test Name*.



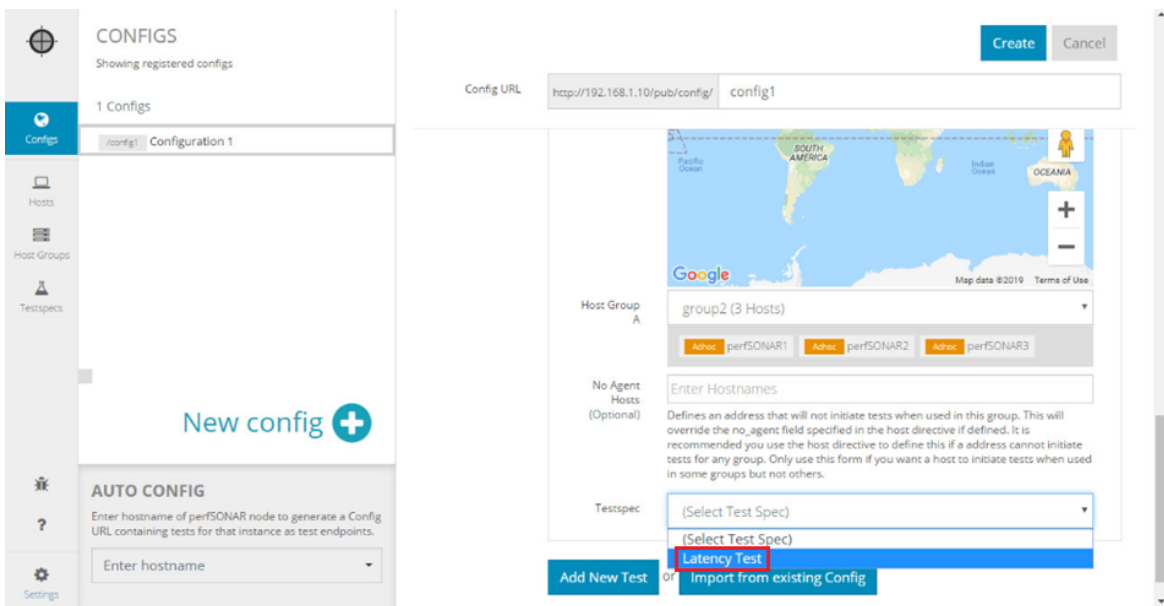
Step 3. Click of *Service Type*, select *Latency* from the list.



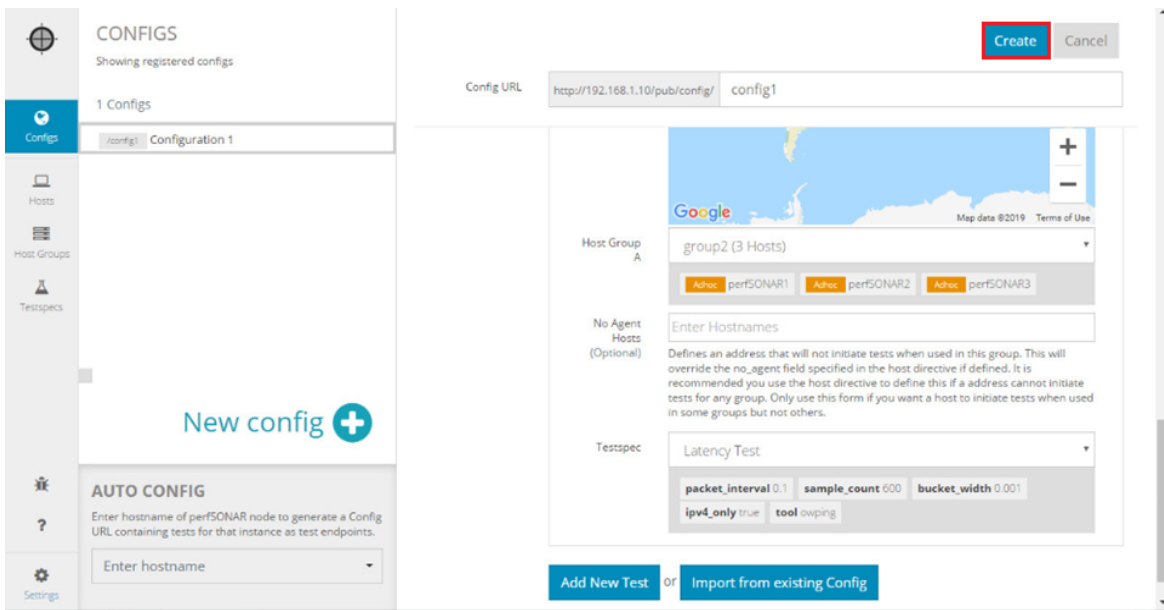
Step 4. Scroll down and click *Host Group A*, a list will be displayed, select *group2*. The user will see the IP addresses of the three nodes involved in the test.



Step 5. Click of *Testspec*, a list will be displayed, select *Latency_Test*.



Step 6. Click on *Create* to save the configuration.



6 Visualizing the measurement data using pScheduler monitor

At this point, the user has created and published a pSConfig file. This file is accessible from the perfSONAR nodes on the topology. In this section the user will run the pSConfig archive using a pScheduler agent. First, the user will login to perfSONAR2 node and add the configuration file. Secondly, the user will see the test schedule using pScheduler monitor.

Step 1. On the topology, click on perfSONAR2 and enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64
perfsonar2 login: admin
Password:
Last login: Mon Apr 8 16:53:24 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
[admin@perfsonar2 ~]$_
```

Step 2. To run the pSConfig template type the following command:

```
sudo psconfig remote add --configure-archives
"https://192.168.1.10:8443/pub/config/config1?format=psconfig"
```

The user will be required to enter the password as `admin`.

```

[admin@perfsonar2 ~]# sudo psconfig remote add --configure-archives "https://192.168.1.10:8443/pub/c
onfig/config1?format=psconfig"
[sudo] password for admin:
=== pScheduler Agent ===
Added remote configuration https://192.168.1.10:8443/pub/config/config1?format=psconfig

=== MaDDash Agent ===
Added remote configuration https://192.168.1.10:8443/pub/config/config1?format=psconfig
[admin@perfsonar2 ~]# _

```

Step 3. After a minute, type the command `pscheduler monitor`. The user will see the scheduled tasks through pScheduler monitor. Notice that the latency task is running twice each hour, and the throughput test once every four hours.

```

2019-04-08T17:01:23-04:00          pScheduler Monitor          perfsonar2
2019-04-08T20:56:47Z Finished    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-08T20:56:46Z Finished    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-08T21:00:49Z Running    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-08T21:00:52Z Running    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-08T21:18:52Z Pending    throughput --duration PT20S --source 192.168.2.10 --ip-version 4+
2019-04-08T22:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-08T22:00:52Z Pending    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-08T23:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-08T23:00:52Z Pending    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-09T00:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-09T00:00:52Z Pending    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-09T01:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-09T01:00:52Z Pending    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-09T01:02:15Z Pending    throughput --duration PT20S --source 192.168.2.10 --ip-version 4+
2019-04-09T01:30:52Z Pending    throughput --duration PT20S --source 192.168.2.10 --ip-version 4+
2019-04-09T02:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-09T02:00:52Z Pending    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-09T03:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-09T03:00:52Z Pending    latency --source 192.168.2.10 --dest 192.168.3.10 --data-ports 8+
2019-04-09T04:00:49Z Pending    latency --source 192.168.2.10 --dest 192.168.1.10 --data-ports 8+
2019-04-09T05:23:31Z Pending    throughput --duration PT20S --source 192.168.2.10 --ip-version 4+
2019-04-09T09:03:44Z Pending    throughput --duration PT20S --source 192.168.2.10 --ip-version 4+

```

Step 4. To exit from pScheduler monitor, press `Ctrl+c`.

This concludes Lab 9.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swany and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709-perfSONAR-11-pScheduler_CLI-v2.pdf.

6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon," [Online]. Available: <https://software.ed.net/esmond/>.



UNIVERSITY OF
SOUTH CAROLINA

PERFSONAR

Lab 10: Configuring pScheduler Limits

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives	3
Lab topology	3
Lab settings.....	3
Lab roadmap.....	4
1 Introduction.....	4
1.1 Overview of perfSONAR limits Files	4
1.2 Identifiers	5
1.3 Classifiers	5
1.4 Rewrite.....	6
1.5 Limits.....	6
1.6 Applications	6
2 Sample limits files	6
3 Applying perfSONAR limits files.....	9
3.1 Testing the first limits configuration file (limits-1.conf).....	9
3.2 Testing second limits configuration file (limits-2.conf)	12
3.3 Testing third limits configuration file (limits-3.conf).....	14
References.....	17

Overview

This lab introduces the reader to pScheduler limits configuration file. The lab describes the components and the syntax of the configuration file how to configure it in order to apply rules to the tests between perfSONAR nodes in the topology.

Objectives

By the end of this lab, the user will:

1. Understand the concept of pScheduler limits.
2. Modify the pScheduler limits configuration file.
3. Verify the impact on pScheduler test after modifying the limits configuration file.

Lab topology

Figure 1 illustrates the topology used for this lab. The topology includes three perfSONAR nodes labeled perfSONAR1, perfSONAR2, perfSONAR3 and a Client host. The perfSONAR nodes run a Linux CentOS 7, and the Client runs a lightweight Linux distribution (*Lubuntu*). The Client host is used to access perfSONAR graphical user interface.

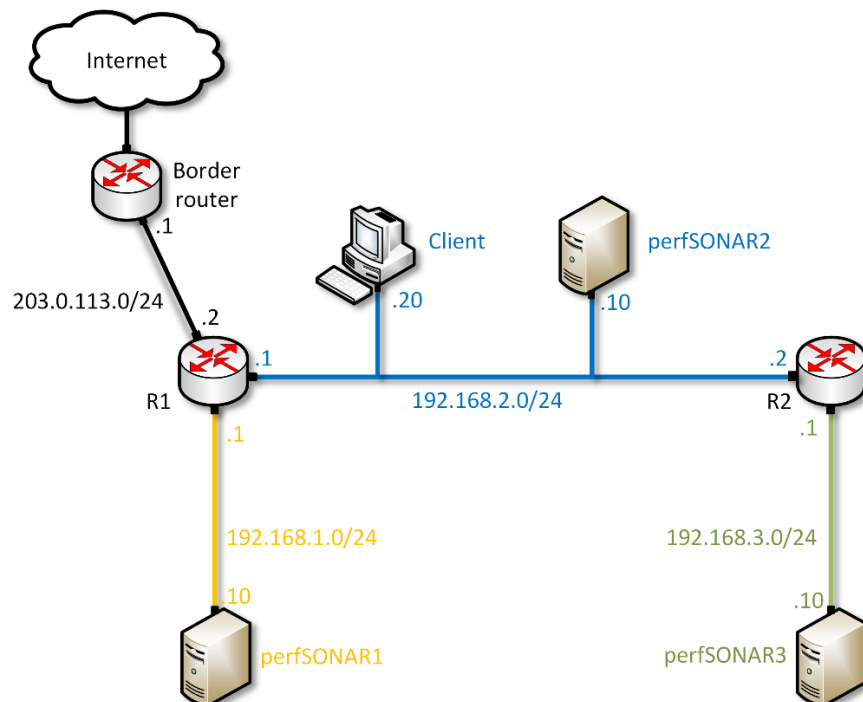


Figure 1. Lab topology.

Lab settings

The information in Table 1 provides the credentials to access to perfSONAR nodes.

Table 1. Credentials to access perfSONAR1, perfSONAR2 and perfSONAR3.

Device	IP Address	Account	Password
perfSONAR1	192.168.1.10	admin	admin
perfSONAR2	192.168.2.10	admin	admin
perfSONAR3	192.168.3.10	admin	admin

Lab roadmap

This lab includes the following tasks:

1. Section 1: Introduction.
2. Section 2: Sample limits files.
3. Section 3: Applying perfSONAR limits files.

1 Introduction

The perfSONAR toolkit provides a detailed framework for network performance measurement across single and multiple network domains¹. An integral component of this solution is the pScheduler tool which is responsible for executing desired network performance tests, also termed as tasks. In particular, pScheduler runs on a server and can either be called using a graphical user interface (GUI), command line input (CLI), or through an application programmer interface (API).

Now perfSONAR users can request a wide range of end-to-end monitoring tests through pScheduler, e.g., such as latency, throughput, loss, etc. However, in general, system and network administrators will want to control various aspects of such user-initiated tests, i.e., as per policy, resource limitations, timing constraints, etc. To accommodate such needs, the perfSONAR framework also provides the ability to control who can run tasks, the types of tasks allowed, and their associated parameter ranges². Specifically, this control is implemented through a limits configuration file. Hence this lab focuses on the contents of this file and how to modify it to achieve desired perfSONAR operation. Note that the default limits configuration file in the perfSONAR installation is also available in the toolkit sources³.

1.1 Overview of perfSONAR limits Files

The perfSONAR limits configuration file uses JavaScript Object Notation (JSON) notation and is named `limits.conf`. This file is located in the `/etc/pscheduler` directory and is read by pScheduler to determine the validity of user task requests. Namely, the pScheduler server checks and applies the latest instance of this file for any new requests and also regularly checks for updates every 15 seconds. Any changes to the `limits.conf` file are also tracked in a log file, generally located in the `/var/log/pscheduler.log` file.

The pScheduler limits file plays a very important role in verifying user requests. Otherwise the pScheduler system will simply accept all requests in case of missing and/or improper `limits.conf` files. As a result, it is important to make sure that the limits file is properly checked for validity before being installed for use by pScheduler. This particular step can also be done through the pScheduler `validate-limits` command, as applied subsequently. Overall, the limits configuration file contains a single JSON object with several attribute-value pairs given by:

```
{
  "#": "Skeletal pScheduler limit configuration",
  "identifiers": [ ... ],
  "classifiers": [ ... ],
  "rewrite": [ ... ],
  "limits": [ ... ],
  "applications": [ ... ]
}
```

1.2 Identifiers

The first part of the verification process is to identify who is actually requesting the performance measurement task. Hence the identifiers component of the limits file contains identifier objects with several attribute fields including *name* (string containing a unique identifier name), *description* (string describing identifier), *type* (string indicating what method to determine if requester should be identified in a category), *data* (JSON object containing *type* object specific data), and *invert* (optional boolean value to invert identification after evaluation). In particular, the *type* and its associated *data* field can support a range of identification strategies, e.g., using hints about the requester, pre-specified classless inter-domain routing (CIDR) IPv4 or IPv6 address lists, downloaded CIDR IP address lists from a website, list of non-bogon IP addresses (to prevent malicious dark web activities), reverse IP address resolution, using jq scripts, or local identifiers.

1.3 Classifiers

The second part of the verification process groups the identifiers into broader categories. Hence the classifiers component of the limits files contains classifier objects with several attribute fields including *name* (string containing a unique identifier name), *description* (string describing identifier), and *identifiers* (string array listing identifiers to be part of classifier). In many cases requesters are classified as friendlies or hostiles.

1.4 Rewrite

The third part of the verification process allows the pScheduler system to make changes to requested tasks. These changes are specified in *rewrite* attribute pair and done using jq scripts. In particular, these scripts call a set functions after initial validation but before limit enforcement (Section 2.4). Overall, the *rewrite* attribute can be used to implement a variety of actions/modifications, e.g., enforce tests from specific interfaces, throttle bandwidth rates, enforce minimum durations, etc.

1.5 Limits

The fourth part of the verification process determines whether or not the requested test parameters fall within acceptable ranges. Hence the limits component of the limits file contains limits objects with several attribute fields including *name* (string containing a unique identifier name), *description* (string describing identifier), *clone* (string naming other limit to be used as a starting point), *type* (type of limit to be checked), *data* (JSON object containing *type*-specific data), and *invert* (optional boolean value to invert identification after evaluation). Note that these attributes are very similar to the identifier's attributes (detailed in Section 2.1). Overall, each limit is individually evaluated and passed or failed. Namely, the *type* field of a limits object can be passed or failed using various strategies, i.e., explicit pass or fail, use of jq scripts to evaluate test parameters, checking of run times or ranges, comparing parameters versus a template of acceptable values, and comparing against a list of acceptable test types.

1.6 Applications

The fifth part of the verification process determines if the test parameters make it permissible, and this check is done by using a set of limit applications. Namely, each classifier is tied to a set of conditions which must be passed. Hence the applications component of the limits files contains *limit application* objects with several attribute fields including *description* (string describing identifier), *classifier* (string naming classifier to which application should be applied), *apply* (array of limit requirements), *invert* (boolean value to invert application result after evaluation), and *stop-on-failure* (boolean value indicating if failure to pass application should prompt further application list evaluations). These application limits are evaluated sequentially, and a task is only run if it passes all of them (or if it passes subsequent application list checks if *stop-on-failure* is false).

2 Sample limits files

Overall, the limits configuration file lets pScheduler exercise a very broad range of control over user task requests. However, providing an exhaustive set of examples to detail every possible verification option is clearly not feasible here. Instead, three sample limit files are used to highlight different components of the pScheduler limits framework. In

particular, these files are located in the `/home/admin` directory and named as `limits-1.conf`, `limits-2.conf`, and `limits-3.conf`. These files are now discussed further.

Step 1. Open perfSONAR2 and enter the username `admin` and password `admin`. Note that the password will not be displayed while typing it.

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64

perfsonar2 login: admin
Password:
Last login: Sun Jan 27 17:27:33 on tty1
Welcome to the perfSONAR Toolkit v4.1.5-1.el7

You may create accounts to manage this host through the web interface by running the following as root:

/usr/lib/perfsonar/scripts/nptoolkit-configure.py

The web interface should be available at:

https://[host address]/toolkit
'abrt-cli status' timed out
[admin@perfsonar2 ~]$_
```

Step 2. On the perfSONAR2 host go to the directory containing the sample limits files by directly typing the command:

```
cd /home/admin

[admin@perfsonar2 ~]$_
[admin@perfsonar2 ~]$_
```

Step 3. To ensure the correct directory, enter the `ls` command to verify that the three sample limits files are listed:

```
[admin@perfsonar2 ~]$_ ls
limits-1.conf limits-2.conf limits-3.conf
[admin@perfsonar2 ~]$_
```

Step 4. Now open the first limits file, `limits-1.conf`, in order to inspect its contents. Type the following command:

```
nano limits-1.conf

[admin@perfsonar2 ~]$_ nano limits-1.conf _
```

This JSON file contains an identifiers component (object array) with a several objects, one of which is an object with the `name` attribute defined as `certain-group`. This particular object is subsequently used to identify the subnet which cannot run tasks on perfSONAR2.

```
"identifiers": [
  {
    "name": "certain-group",
    "description": "Requests coming from 192.168.0.0/16",
    "type": "ip-cidr-list",
    "data": {
      "cidrs": [ "192.168.0.0/16" ]
    }
  }
]
```

Additionally, the *limits-1.conf* file also contains a classifiers component (object array) with an object whose *name* attribute is defined as *hostiles*. This particular object also has an *identifiers* attribute defined as (the above-defined) "certain-group", i.e., to prevent identifiers in this group from running tasks on perfSONAR2.

```
"classifiers": [
  {
    "name": "hostiles",
    "description": "Identifiers we find unfriendly",
    "identifiers": [ "certain-group" ]
  }
]
```

Step 5. To close the current file type `Ctrl+x`. Next, open the second limits file, *limits-2.conf*, by typing the following command:

```
nano limits-2.conf
```

```
[admin@perfsonar2 ~]# nano limits-2.conf _
```

This file contains an identifiers component (object array) with two objects with their *name* attributes set to *perfSONAR1* and *perfSONAR2*, respectively. Specifically, both of these objects define their *type* attributes as *ip-cidr-list* and related *data* attributes as a *cidrs* object array (containing two IP addresses, i.e., *192.168.1.10* and *192.168.3.10*).

```
"identifiers": [
  {
    "name": "perfSONAR1",
    "description": "Requests coming from perfSONAR1",
    "type": "ip-cidr-list",
    "data": {
      "cidrs": [ "192.168.1.10" ]
    }
  },
  {
    "name": "perfSONAR3",
    "description": "Requests coming from perfSONAR3",
    "type": "ip-cidr-list",
    "data": {
      "cidrs": [ "192.168.3.10" ]
    }
  }
]
```

Using the above, the *limits-2.conf* file also includes a classifier component (object array) with two objects with their *name* attributes set to *friendlies* and *hostiles*, respectively. Specifically, the first object contains an *identifiers* attribute listing IP address(es) that can send requests to perfSONAR2 (in this case only perfSONAR3 is allowed). Meanwhile, the second object contains an *identifiers* attribute listing IP address(es) who cannot send requests to perfSONAR2 (in this case only perfSONAR1 is not allowed).


```

"classifiers": [
  {
    "name": "friendlies",
    "description": "Identifiers we find friendly",
    "identifiers": [ "perfSONAR3" ]
  },
  {
    "name": "hostiles",
    "description": "Identifiers we find unfriendly",
    "identifiers": [ "perfSONAR1" ]
  }
]

```

Step 6. Finally, exit the *nano* editor as before by typing `Ctrl+x`. Then open the third limits file, *limits-3.conf*, by typing the following command:

```
nano limits-3.conf
```

```
[admin@perfsonar2 ~]$ nano limits-3.conf _
```

This file contains a limits component (object array) with a single object with the *name* attribute set to *throughput-default-time*. Furthermore, the *data* attribute here defines another object with a *limit* attribute (object) specifying the upper and lower run time durations for throughput tasks, i.e., 5 seconds and 15 seconds, respectively.

```

"name": "throughput-default-time",
"description": "Throughput time limits",
"type": "test",
"data": {
  "test": "throughput",
  "limit": {
    "duration": {
      "range": {
        "lower": "PT5S",
        "upper": "PT15S"
      }
    }
  }
}

```

Step 7. Exit the *nano* editor as before by typing `Ctrl+x`.

3 Applying perfSONAR limits files

The three sample limits files detailed in Section 3 are now used to test and verify the capabilities of pScheduler limits framework.

3.1 Testing the first limits configuration file (limits-1.conf)

Step 1. In perfSONAR2, go to the local file directory containing the three sample limit configuration files typing the following command:

```
cd /home/admin
```

```
[admin@perfsonar2 ~]# cd /home/admin
[admin@perfsonar2 ~]# _
```

Step 2. Type the command `ls` to ensure that all three template files are listed in the directory.

```
[admin@perfsonar2 ~]# ls
limits-1.conf limits-2.conf limits-3.conf
[admin@perfsonar2 ~]# _
```

Step 3. The pScheduler limits file is located in the local directory `/etc/pscheduler` and called `limits.conf`, i.e., `/etc/pscheduler/limits.conf`. Now pScheduler can only recognize a limits file with this particular name. Hence in order to test the first limit configuration file, `limits-1.conf` must be copied from `/home/admin` and renamed into the `/etc/pscheduler` directory. To do this, type the command in the `/home/admin` directory, i.e., to create a renamed copy:

```
cp limits-1.conf limits.conf
```

```
[admin@perfsonar2 ~]# cp limits-1.conf limits.conf
[admin@perfsonar2 ~]# _
```

Step 4. To verify that the copy succeeded, type `ls`.

```
[admin@perfsonar2 ~]# ls
limits-1.conf limits-2.conf limits-3.conf limits.conf
[admin@perfsonar2 ~]#
```

Step 5. Next, move the copied `limits.conf` file to the pScheduler directory `/etc/pscheduler`. To do this, type the command shown below and enter the password `admin`.

```
sudo mv limits.conf /etc/pscheduler
```

```
[admin@perfsonar2 ~]# sudo mv limits.conf /etc/pscheduler
[sudo] password for admin:
[admin@perfsonar2 ~]# _
```

Step 6. To ensure the file was properly moved, type `ls` to make sure that the `limits.conf` is no longer in the current `/home/admin` directory:

```
[admin@perfsonar2 ~]# ls
limits-1.conf limits-2.conf limits-3.conf
[admin@perfsonar2 ~]# _
```

Step 7. Now return to the main directory by entering the following command:

```
cd ~
```

```
[admin@perfsonar2 ~]# cd ~
[admin@perfsonar2 ~]#
```

Step 8. Since the pScheduler *limits.conf* file has just been changed, it is important to verify that that it is still valid by using the pScheduler `validate-limits` command. To verify this, type the command shown below and note the output. If the configuration is valid, the output should display *Limit configuration is valid*.

```
pscheduler validate-limits
```

```
[admin@perfsonar2 ~]$ pscheduler validate-limits
Limit configuration is valid.
[admin@perfsonar2 ~]$
```

Step 9. As detailed in Section 3, the *limits-1.conf* file is designed to block perfSONAR1 and perfSONAR3 from running tasks on perfSONAR2. However, these limits should not prevent perfSONAR1 from running a task on perfSONAR3 or vice versa. In order to test these restrictions, try to schedule a throughput task to perfSONAR2 by typing command shown below. This task will fail.

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10
```

```
[admin@perfsonar2 ~]$ pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10
Submitting task...
Unable to post task: Error while tasking 192.168.2.10: Unable to post task to 192.168.2.10: Task forbidden by limits:
Hints:
  requester: 192.168.1.10
  server: 192.168.2.10
Identified as everybody, certain-group
Classified as default, hostiles
Application: Hosts we don't want running any tests
  Group 1: Limit 'never' failed: Forced failure
  Group 1: Want all, 0/1 passed, 1/1 failed: FAIL
  Group 1: Failed; stopping here.
Application FAILS
  Failed - Stop Forced
Proposal does not meet limits
The 'pscheduler troubleshoot' command may be of use in problem
diagnosis. 'pscheduler troubleshoot --help' for more information.
[admin@perfsonar2 ~]$ _
```

Step 10. Next, verify that the rule also applies to perfSONAR3 by typing the command shown below. This task request will also fail.

```
pscheduler task throughput --source 192.168.3.10 --dest 192.168.2.10
```

```

[admin@perfsonar2 ~]$ pscheduler task throughput --source 192.168.3.10 --dest 192.168.2.10
Submitting task...
Unable to post task: Error while tasking 192.168.2.10: Unable to post task to 192.168.2.10: Task forbidden by limits:
Hints:
  requester: 192.168.3.10
  server: 192.168.2.10
Identified as everybody, certain-group
Classified as default, hostiles
Application: Hosts we don't want running any tests
  Group 1: Limit 'never' failed: Forced failure
  Group 1: Want all, 0/1 passed, 1/1 failed: FAIL
  Group 1: Failed; stopping here.
Application FAILS
  Failed - Stop Forced
Proposal does not meet limits
The 'pscheduler troubleshoot' command may be of use in problem diagnosis. 'pscheduler troubleshoot --help' for more information.
[admin@perfsonar2 ~]$ _

```

Step 11. Finally, verify that perfSONAR1 and perfSONAR3 can still schedule tasks between themselves. To verify this, type the command shown below. This task request should succeed and display throughput results.

```
pscheduler task throughput --source 192.168.3.10 --dest 192.168.1.10
```

```

[admin@perfsonar2 ~]$ pscheduler task throughput --source 192.168.3.10 --dest 192.168.1.10
Submitting task...
Task URL:
https://192.168.3.10/pscheduler/tasks/733544b3-3c6a-4430-8396-66c996d472a1
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.3.10/pscheduler/tasks/733544b3-3c6a-4430-8396-66c996d472a1/runs/4446b3d4-1416-44c8-b6ec-b6c867b1a738
Starts 2019-04-22T15:53:52Z (~5 seconds)
Ends 2019-04-22T15:54:12Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput      Retransmits      Current Window
0.0 - 1.0     7.04 Gbps       292              878.94 KBytes
1.0 - 2.0     5.75 Gbps       217              632.78 KBytes
2.0 - 3.0     5.51 Gbps       454              702.28 KBytes
3.0 - 4.0     6.59 Gbps       136              1.07 MBytes
4.0 - 5.0     6.35 Gbps       137              807.98 KBytes
5.0 - 6.0     6.73 Gbps       390              1.06 MBytes
6.0 - 7.0     6.10 Gbps       98               848.53 KBytes
7.0 - 8.0     5.66 Gbps       253              676.22 KBytes
8.0 - 9.0     5.95 Gbps       252              654.50 KBytes
9.0 - 10.0    6.42 Gbps       0                1.54 MBytes

Summary
Interval      Throughput      Retransmits
0.0 - 10.0    6.21 Gbps       2229

No further runs scheduled.
[admin@perfsonar2 ~]$ _

```

3.2 Testing second limits configuration file (limits-2.conf)

Step 1. In perfSONAR2, go to the local file directory containing the 3 sample limit configuration files. Namely, type the command shown below.

```
cd /home/admin
```

```
[admin@perfsonar2 ~]# cd /home/admin
[admin@perfsonar2 ~]# _
```

Step 2. Copy the contents of *limits-2.conf* into a file titled *limits.conf* by entering the following command:

```
cp limits-2.conf limits.conf
```

```
[admin@perfsonar2 ~]# cp limits-2.conf limits.conf
[admin@perfsonar2 ~]# _
```

Step 3. Move this new limit file into the */etc/pscheduler* directory by entering the command. Enter the password `admin` when prompted.

```
sudo mv limits.conf /etc/pscheduler
```

```
[admin@perfsonar2 ~]# sudo mv limits.conf /etc/pscheduler
[sudo] password for admin:
[admin@perfsonar2 ~]# _
```

Step 4. Now return to the main directory again by entering the following command:

```
cd ~
```

```
[admin@perfsonar2 ~]# cd ~
[admin@perfsonar2 ~]#
```

Step 5. Check the validity of the limits file once again by using the command shown below. The output should read limit configuration is valid.

```
pscheduler validate-limits
```

```
[admin@perfsonar2 ~]# pscheduler validate-limits
Limit configuration is valid.
[admin@perfsonar2 ~]#
```

Step 6. As detailed in Section 3 the *limits-2.conf* file is designed to block any requests from perfSONAR1 but allow requests from perfSONAR3. In order to test these restrictions, try to schedule a throughput task to perfSONAR2 by typing the command shown below. This task request should fail based upon the specified limits.

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10
```

```

[admin@perfsonar2 ~]$ pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10
Submitting task...
Unable to post task: Error while tasking 192.168.2.10: Unable to post task to 192.168.2.10: Task forbidden by limits:
Hints:
  requester: 192.168.1.10
  server: 192.168.2.10
Identified as everybody, certain-group
Classified as default, hostiles
Application: Hosts we don't want running any tests
  Group 1: Limit 'never' failed: Forced failure
  Group 1: Want all, 0/1 passed, 1/1 failed: FAIL
  Group 1: Failed; stopping here.
Application FAILS
  Failed - Stop Forced
Proposal does not meet limits
The 'pscheduler troubleshoot' command may be of use in problem diagnosis. 'pscheduler troubleshoot --help' for more information.
[admin@perfsonar2 ~]$ _

```

Step 7. Next, verify that perfSONAR3 is still able to run tasks to perfSONAR2. Specifically, type the command shown below. This task request should succeed and display throughput results.

```
pscheduler task throughput --source 192.168.3.10 --dest 192.168.2.10
```

```

[admin@perfsonar2 ~]$ pscheduler task throughput --source 192.168.3.10 --dest 192.168.2.10
Submitting task...
Task URL:
https://192.168.3.10/pscheduler/tasks/0fb58345-4cd7-4cc2-b5a8-d4fba664cd43
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.3.10/pscheduler/tasks/0fb58345-4cd7-4cc2-b5a8-d4fba664cd43/runs/8e9f2882-2ad8-46ba-a9b8-1256eb59caab
Starts 2019-04-22T16:55:57Z (~7 seconds)
Ends 2019-04-22T16:56:16Z (~18 seconds)
Waiting for result...

* Stream ID 5
Interval      Throughput    Retransmits   Current Window
0.0 - 1.0    7.96 Gbps     629           422.82 KBytes
1.0 - 2.0    5.76 Gbps     129           459.02 KBytes
2.0 - 3.0    7.40 Gbps     386           729.79 KBytes
3.0 - 4.0    6.88 Gbps     177           651.68 KBytes
4.0 - 5.0    6.90 Gbps     185           637.12 KBytes
5.0 - 6.0    7.05 Gbps     289           832.68 KBytes
6.0 - 7.0    7.13 Gbps     254           434.48 KBytes
7.0 - 8.0    7.61 Gbps     0             1.03 MBytes
8.0 - 9.0    7.32 Gbps     234           505.35 KBytes
9.0 - 10.0   7.36 Gbps     177           828.26 KBytes

Summary
Interval      Throughput    Retransmits
0.0 - 10.0    7.14 Gbps     2380

No further runs scheduled.
[admin@perfsonar2 ~]$ _

```

3.3 Testing third limits configuration file (limits-3.conf)

Step 1. In perfSONAR2, go to the local file directory containing the 3 sample limit configuration files. Type the following command.

```
cd /home/admin
```

```
[admin@perfsonar2 ~]# cd /home/admin
[admin@perfsonar2 ~]# _
```

Step 2. Copy the contents of `limits-3.conf` into a file named `limits.conf` by entering the following command.

```
cp limits-3.conf limits.conf
```

```
[admin@perfsonar2 ~]# cp limits-3.conf limits.conf
[admin@perfsonar2 ~]# _
```

Step 3. Move this new limit file into the `/etc/pscheduler` directory by entering the command shown below. Enter the password `admin` when prompted.

```
sudo mv limits.conf /etc/pscheduler
```

```
[admin@perfsonar2 ~]# sudo mv limits.conf /etc/pscheduler
[sudo] password for admin:
[admin@perfsonar2 ~]# _
```

Step 4. Now return to the main directory again by entering the following command:

```
cd ~
```

```
[admin@perfsonar2 ~]# cd ~
[admin@perfsonar2 ~]#
```

Step 5. Check the validity of the limits file once again by using the command shown below. The output should read *Limit configuration is valid.*

```
pscheduler validate-limits
```

```
[admin@perfsonar2 ~]# pscheduler validate-limits
Limit configuration is valid.
[admin@perfsonar2 ~]#
```

Step 6. As detailed in Section 3, the `limits-3.conf` file is designed to restrict the maximum duration of throughput tasks to 15 seconds. Hence any request to schedule such a task for 20 seconds should fail. In order to test this restriction, try to schedule a 20 second throughput task to perfSONAR2 by typing in the command shown below. This task request should fail based upon the specified duration limit. In perfSONAR1, type the following command:

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10 -t 20
```

```

[admin@perfsonar1 ~]# pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10 -t 20
Submitting task...
Unable to post task: Error while tasking 192.168.2.10: Unable to post task to 192.168.2.10: Task forbidden by limits:
Hints:
  requester: 192.168.1.10
  server: 192.168.2.10
Identified as everybody
Classified as default
Application: Defaults applied to non-friendly hosts
  Group 1: Limit 'innocuous-tests' failed: Passed but inverted
  Group 1: Limit 'throughput-default-time' failed: Duration not in PT5S..PT15S
  Group 1: Limit 'idleex-default' failed: Test is not 'idleex'
  Group 1: Want any, 0/3 passed, 3/3 failed: FAIL
  Group 1: Failed; stopping here.
Application FAILS
Proposal does not meet limits
The 'pscheduler troubleshoot' command may be of use in problem diagnosis. 'pscheduler troubleshoot --help' for more information.
[admin@perfsonar1 ~]# _

```

Step 7. Meanwhile, a request to schedule throughput tasks with durations of 15 seconds or less should be successful. In order to test this bound, try to schedule a 15 second throughput task to perfSONAR2 by typing in the command shown below. This task request should succeed and display throughput results.

```
pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10 -t 15
```

```

[admin@perfsonar2 ~]# pscheduler task throughput --source 192.168.1.10 --dest 192.168.2.10 -t 15
Submitting task...
Task URL:
https://192.168.1.10/pscheduler/tasks/29b2da77-d7e5-4a50-bd33-f0332ded84b2
Running with tool 'iperf3'
Fetching first run...

Next scheduled run:
https://192.168.1.10/pscheduler/tasks/29b2da77-d7e5-4a50-bd33-f0332ded84b2/runs/8e97bf84-6671-4a72-ac25-5d8ae5e5a562
Starts 2019-06-23T21:27:47Z (~8 seconds)
Ends 2019-06-23T21:28:11Z (~23 seconds)
waiting for result...

* Stream ID 5
Interval      Throughput      Retransmits      Current Window
0.0 - 1.0     7.94 Gbps       51                1.30 MBytes
1.0 - 2.0     6.95 Gbps       264               457.57 KBytes
2.0 - 3.0     6.82 Gbps       15                527.07 KBytes
3.0 - 4.0     6.49 Gbps       141               803.64 KBytes
4.0 - 5.0     6.91 Gbps       0                 949.89 KBytes
5.0 - 6.0     6.98 Gbps       131               826.81 KBytes
6.0 - 7.0     7.02 Gbps       0                 932.51 KBytes
7.0 - 8.0     6.81 Gbps       293               417.02 KBytes
8.0 - 9.0     6.86 Gbps       0                 851.42 KBytes
9.0 - 10.0    6.56 Gbps       147               787.71 KBytes
10.0 - 11.0   6.24 Gbps       131               816.67 KBytes
11.0 - 12.0   6.51 Gbps       241               398.20 KBytes
12.0 - 13.0   6.94 Gbps       0                 855.77 KBytes
13.0 - 14.0   6.79 Gbps       221               764.54 KBytes
14.0 - 15.0   6.88 Gbps       24                874.59 KBytes

Summary
Interval      Throughput      Retransmits
0.0 - 15.0    6.85 Gbps       1659

No further runs scheduled.
[admin@perfsonar2 ~]#

```

This concludes Lab 10.

References

1. NSRC, "What is perfSONAR?," [Online]. Available: <https://learn.nsrc.org/perfsonar/what-is-perfsonar>.
2. B. Tierney, J. Metzger, E. Boyd, A. Brown, R. Carlson, M. Zekau, J. Zurawski, M. Swamy and M. Grigoriev, "perfSONAR: instantiating a global network measurement," in SOSP workshop, Real overlays and distributed systems.
3. How to use the linux traffic control panagiotis vouzis," [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control/>.
4. perfSONAR Project, "Creating and managing tasks," [Online]. Available: https://docs.perfsonar.net/pscheduler_client_tasks.html.
5. perfSONAR Project, "The pScheduler command-line interface," [Online]. Available: https://www.perfsonar.net/media/medialibrary/2017/09/22/201709perfSONAR-11-pScheduler_CLI-v2.pdf.
6. M. Feit, "CLI user's guide," [Online]. Available: <https://github.com/perfsonar/pscheduler/wiki/CLI-User%27s-Guide>.
7. ESnet, "esmond: ESnet monitoring daemon". Available: <https://software.ed.net/esmond/>.