DisNETPerf Documentation

Release 1.0

Sarah Wassermann

Jun 03, 2019

Contents

DisNETPerf - a Distributed Internet Paths Performance Analyzer - is a tool that allows one to locate the closest RIPE Atlas box (in terms of minimum RTT) to a given IP. You can either run DisNETPerf for a single IP or for a set of IPs. Furthermore, once the closest RIPE Atlas box has been located, DisNETPerf permits to launch traceroutes from this box to a destination IP address provided by the user.

RIPE Atlas is a large active measurement network composed of geographically distributed probes used to measure Internet connectivity and reachability.

How DisNETPerf works:

Given a certain server, with IP address IPs, and a target customer, with IP address IPd, DisNETPerf locates the closest RIPE Atlas probe to IPs, namely IPc, and periodically runs traceroute from IPc to IPd, collecting different path performance metrics such as RTT per hop, end-to-end RTT, etc. The collected data is then used to troubleshoot ''reverse" paths, from the server to the target customer. To select IPc, DisNETPerf makes use of a combined topological and latency-based approach, using standard pings and BGP routing tables. In a nutshell, it locates the RIPE Atlas probe with minimum RTT to the selected server IPs, among a set of prefiltered IPc candidates, which are located at either the same AS of IPs or in the neighbor ASes.

Contents of the documentation:

CHAPTER 1

Installation instructions

To run DisNETPerf, Python 2.7 must be installed. You can download Python 2.7 on <https://www.python.org/download/releases/2.7/> Furthermore, please fulfill all the prerequisites for the RIPE Atlas Toolbox explained on <https://github.com/pierdom/atlas-toolbox>

DisNETPerf has been tested on Debian 7 with Python 2.7.9.

CHAPTER 2

find_psbox.py

This script locates the closest RIPE Atlas probes (called *proximity services boxes*) to IP addresses.

In order to launch this script, please use the following command:

```
python find psbox.py -k <API-KEY> [-n <IP filename>] [-o <targetIP>] [-r {0,1}]
```

<API-Key> points to a RIPE Atlas API Key with *Measurement creation* permissions. Such a key can easily be created through the Web interface of RIPE Atlas.
<IP filename> refers to the name of the file in which the IP addresses DisNETPerf should locate the closest RIPE Atlas box to are listed. **This file has to be stored in the 'input' folder.** The file should contain one IP per line. An IP should be in the usual format, i.e. X.X.X.X where X is an integer >= 0.

If you want to analyze only a single IP, you can also use the -o parameter and replace <targetIP> with that particular IP.

If you set the -r parameter to 1, the recovery-mode will be enabled. For further information about this mode, please have a look at the documentation about it. (The default-value for this parameter is 0.)

2.1 Output

The output file of the script `find_psbox.py` contains information about the targetIPs and the corresponding computed closest boxes. The naming scheme of such an output-file is `<timestamp>_psbox.txt` where `<timestamp>` refers to the timestamp indicating when `find_psbox.py` was launched. This file is saved into the folder *output*.

The lines of an output-file follow the format:

<targetIP> <psBox ID> <psBox IP> <AS number> <min RTT> <label>

where <psBox ID/IP> refers to the ID/IP of the found proximity service box, <AS number> to the AS number in which this probe is installed, and <min RTT> to the minimum RTT measured from this box to the targetIP.
Finally, <label> can have the following values:

- **[OK]**: the candidate RIPE Atlas boxes (i.e. among the ones the closest one has been chosen) are either in the same AS as <target-IP> or in the neighbour-ASes
- **[NO_AS]**: <targetIP> could not be mapped to an AS and thus the candidate RIPE Atlas boxes have been chosen randomly
- **[Random]**: No candidate boxes have been found in the same AS as <target-IP> and in the neighbour-ASes. Boxes have thus been chosen randomly

Please note that the lines in this file are tab-separated.

CHAPTER 3

launch_traceroutes.py

This script allows you to launch traceroutes from RIPE Atlas probes towards a given destination IP.
To execute this script, use the following command:

```
python launch traceroutes.py -k <API-Key> [-n <IP filename>] [-o <targetIP>] -d <dest␣
↪IP> [-b <psBox ID>] -f {0,1} [-m <nb traceroutes>] [-t <interval>] [-s <starttime>]␣
↪[-p <stoptime>]
```

where:

- -k <API-Key>: points to a RIPE Atlas API Key with *Measurement creation* permissions. Such a key can easily be created through the Web interface of RIPE Atlas
- -n <IP filename>: the name of the file containing the IP addresses which DisNETPerf should locate the closest probe for. **This file has to be stored in the 'input' folder**
- -o <targetIP>: the IP address which DisNETPerf should locate the proximity service box for
- -d <dest IP>: the IP address the traceroute measurements are issued to
- -b <psBox ID>: the ID of the probe the measurements should be launched from
- -f {0, 1}: if set to 1, the proximity service box has to be already identified. If the parameter -n is used, the IDs of the boxes have to be indicated in the file; if -o is set, the ID has to be provided through the parameter -b. If the -f parameter is equal to 0, DisNETPerf first looks for the corresponding proximity service boxes before launching the traceroute measurements
- -m <nb traceroutes>: the number of measurements to launch
- -t <interval>: the time between two consecutive measurements, in seconds
- -s <starttime>: the UNIX timestamp indicating when the first measurement should be issued
- -p <stoptime>: the UNIX timestamp indicating when the last measurement should be launched

Please note that, when you want to directly indicate the probes to be used within the file containing the different IP addresses, each line contains the IP address and the ID of the corresponding box (tab-separated). If you want to locate the closest box first, each line should only contain an IP address.

Moreover, DisNETPerf is quite flexible when it comes to the parameters -m, -t , and -s. Indeed, the following combinations are possible:

m	t	s	behaviour
X	X	X	launch now one measurement (i.e. a on/off measurement)
O	O	X	launch measurement now with stoptime = m * t
O	X	NA	launch measurement now (or at the indicated starttime) and use the default interval (= 600 secs)

('X': parameter not used; 'O': parameter used; 'NA': don't care)

CHAPTER 4

get_traceroute_results.py

This script retrieves and saves the results of traceroute measurements.

To launch it, please use the command

```
python get_traceroute_results.py -n <UDM filename>
```

<UDM filename> refers to the filename of the file in which the measurement IDs of the measurements are stored. **This file has to be stored in the 'input' folder.** Each line has to follow this format:

```
<UDM_ID_1>...<UDM_ID_X> <targetIP
```

where <UDM_ID_1>...<UDM_ID_X> are the IDs of the traceroute measurements launched towards the IP address <targetIP>

4.1 Output

get_traceroute_results.py creates a file in the folder *output*. The naming scheme of such an output-file is `<timestamp>_scheduled_traceroutes.txt` where `<timestamp>` refers to the timestamp indicating when `get_traceroute_results.py` was launched. In this file, each measurement is represented in the following way:

```
PROBEID: <ID>
TIMESTAMP: <TS>
```

```
NBHOPS: <N>
HOP: <IP 1> [<avg RTT>]
...
HOP: <IP N> [<avg RTT>]
ASPATH: <ASHOP 1>...<ASHOP X>
POPPATH: <POPHOP 1>...<POPHOP Y>
IPPATH: <IPHOP 1>...<IPHOP Z>
```

<IP X> either reports the IP address of the encountered router interface or is replaced by NA_TR. NA_TR indicates that the IP address of the router could not be inferred (and therefore the average RTT could not be computed). ASPATH, POPPATH, and IPPATH indicate the paths at the three considered levels. Regarding the AS hops, we display NA_MAP when the IP address could not be mapped to an AS, and NA_TR when the IP address is unknown.

CHAPTER 5

Recovery-mode

DisNETPerf offers a recovery-mode when computing the closest RIPE Atlas boxes. If DisNETPerf encounters an internal problem while `find_psbox.py` is running, and stops, you can simply rerun the script and set the parameter -r to 1. DisNETPerf will then automatically detect the created output-file for the failed measurements (if a file has been created) and will launch the remaining measurements and analyses.

For this mode to work properly, some constraints have to be respected:

- the run of find_psbox.py that should perform the recovery must be launched immediately after the failed run. If you run find_psbox.py for another purpose before the recovery-run, the recovery will not be possible anymore
- the created files 'current_ping_measurementIDs.log' and 'ID_To_AS.log' in the folder 'log' are necessary. Please do not delete or modify them.