Visualizaing the Internet Topology

Anshul Sangrame (CS21BTECH11004)
Gautam Singh (CS21BTECH11018)
Varun Gupta (CS21BTECH11060)

Contents

This document contains a report on the Python library `netvis` created by the authors, to visualize the topology of the internet. It covers some implementation details, challenges in implementation and conclusions from using this library.

1 Implementation

`netvis` is a `python` package, installable using `pip`. In the following subsections, we describe the installation and sailent features provided by this library.

The entire source code is hosted on GitHub. The verbose implementation details are provided as comments in the source code, which have been omitted here for brevity.

1.1 Installation

`netvis` can be installed by typing the following in a terminal window.

```
$ git clone https://github.com/goats-9/
   cs3530-assignments
$ cd cs3530-assignments/netvis
$ pip install -e .
```

It is *highly recommended* to install the package in a Python virtual environment.

1.2 Requirements

To help in the visualization of the topology of the Internet, `netvis` depends on the following Python packages.

1) `pandas`. To process the raw data obtained for visualization.
2) `openpyxl`. To handle raw data in Excel files.
3) `pyvis`. To visualize the network in the form of a graph.
4) flask. To integreate the network created and add other features such as a legend for easy viewing and usage.

Additionally, note that this library is suitable for Linux systems only, and also requires the executable `mtr` to be installed on the system.

1.3 Features

The sailent features of `netvis` are the following.

1) A `NetGraph` object to abstract the topology of the Internet into a directed graph, containing a list of nodes and edges.
2) Use of `mtr` (short for *my traceroute*) with appropriate arguments to obtain the entire sequence of hops from source to destination, along with AS numbers of ISPs (if available) in CSV form for easy data handling.
3) Provisions to save and load `NetGraph` objects to and from Excel files so that they can be used in other Python scripts.
4) Provision to create a union of two `NetGraph` objects so that the topology from multiple sources to multiple destinations is *integrated*.
5) Rendering of the graph depicting the topology in HTML, along with the use of `flask` to add a legend and host the entire page on a server. The graph is present in `graph.html`, but the integrated legend is displayed on running the `flask` webserver and going to the specified URL in a browser.

1.4 Simulation

1) The data from each source was collected by running the script

```
$ python3 tests/data_collector.py
```

2) The data from various sources was combined and the graph created by running the script

```
$ python3 tests/visualize.py
```

3) The graph was displayed on running the server as follows.

```
$ cd server
$ flask run
```

2 Challenges

The authors wanted to automate the entire process of visualizing the internet topology. The following challenges were faced in the process.

1) Whenever `netvis` is imported, a dataset about 30 MB in size is downloaded and loaded into a `DataFrame` object. This may not be suitable for systems with constraints on resources like memory or internet speed (such as embedded systems).
   A more suitable solution could have been hosting the data on a webserver and making HTTP GET requests to a URL.
2) `pyvis` does not have a provision to provide a legend for easy use of the generated graph. The authors worked around this using `flask` and adding the legend through a `.csv` file.
3) The outputs of `mtr` did not include the source IP address. It was manually added by making use of the Python standard library module `socket`.

3 Observations

The following conclusions can be made based on the data collected.

1) The graph obtained is a directed acyclic graph, and there are no cycles in the network observed. Oherwise, a packet would travel in a cycle without reaching its intended destination and in turn get dropped. Presence of such a cycle may mean that the routing and forwarding algorithms in that part of the network are bugged.
2) From the raw data, sites hosted in countries east of India have lesser RTTs (less than 200 ms) compared to sites hosted in countries to the west of India (at least 200 ms), when performing traceroutes from sources in India.
3) The opposite is observed when performing traceroutes from sources in Europe (source IP `172.31.49.238`). That is, sites hosted in Oceania had higher average RTT compared to sites hosted in America.
4) a) Using the LAN service available in IITH (source IP `10.5.80.104`), the packets always travelled through the ISP at IITH and sometimes through the NKN Core Network.
   b) Using mobile data, the packets always went through the service providers to which the SIM cards were registered.
   c) Using a personal WiFi network (not in campus, source IP `192.168.1.38`), the packets went through CtrlS datacenters, which might mean that the local ISP uses the services of this datacenter to host and provide Internet services to their customers.
5) Some of the largest ISPs observed in terms of number of hops are
   a) `AS8075`. This corresponds to the datacenter of Microsoft Corporation.
   b) `AS1299`. This corresponds to Arelion, which is a Swedish telecommuniations services provider.
   c) `AS174`. This corresponds to Cogent Communications, which is a multinational internet service provider based in the USA.

4 Resources

1) The raw and processed data is present in the `data` directory.
2) The scripts used to collect and visualize the data are present in the `tests` directory.
3) The `flask` server script is present in the `server` directory.
4) The ASN dataset `data/asn.tsv` was taken and edited from `asntool.com`.
5) The hatch build system was used to package the code.