BGP Measurements Project

Table of Contents

Introduction

This project is to be done in the class VM where the BGPStream libraries are installed. Your code will need to run without modification in the course VM.

In this project we will use the BGPStream tool and its Python interface PyBGPStream to understand the BGP protocol and interact with BGP data. The goal is to gain a better understanding of BGP and to experience how researchers, practitioners and engineers have been using BGPStream to gain insights. More specifically we will be using a newly developed tool that gives us the option to browse both real-time BGP data as well as historical BGP data.

This project has both a report component and a code component. You will need to complete the functions in the `bgpm.py` file and create a single PDF file called `report.pdf` for your submission. These two files (and only these two files) will be zipped and submitted to Canvas.

Folder Structure

Your code submission will be auto-graded in the class VM. The .zip file accompanying this assignment has the folder structure that will be used for grading. It is as follows:

```
rib_files/
update_files/
update_files_blackholing/
bgpm.py
```

`bgpm.py` is the only file that you should modify. This file, along with `report.pdf`, will be zipped and submitted to Canvas. Before submitting your assignment, make sure all your code works with this folder structure on the course VM.

Required Background

The tool we will be using is called BGPStream. It is an open-source tool that provides access to historical and real-time BGP data.

Read the Paper

A high-level overview about how the BGPStream tool was developed was published by the Center for Applied Internet Data Analysis (CAIDA). The paper, *BGPStream: A Software Framework for Live and Historical BGP Data Analysis*, can be found here: https://www.caida.org/publications/papers/2016/bgpstream/bgpstream.pdf

This paper provides useful background and practical examples of BGPStream - be sure you read it.

Run Example Code Snippets

For the following tasks, we require that you use the Python interface of BGPStream (PyBGPStream). You are strongly encouraged to browse the following resources to familiarize yourself with the tool and see example code snippets:

- PyBGPStream API: https://bgpstream.caida.org/docs/api/pybgpstream
- PyBGPStream API Tutorial: https://bgpstream.caida.org/docs/tutorials/pybgpstream
- PyBGPStream Repository: https://github.com/CAIDA/pybgpstream
- Official Examples: https://github.com/CAIDA/pybgpstream/tree/master/examples

Familiarize Yourself with the BGPStream Record Format

The BGPReader command-line tool provides access to BGP records. Here we will show sample command line output from BGPReader for illustration purposes only. In this assignment, you will not be required to use BGPReader or interact with records in this format.

Updates

The box below contains an example of an update record. In the record, the "|" character separates different fields. In yellow we have highlighted the **type** (A stands for Advertisement), the **advertised prefix** (210.180.224.0/19), **the path** (11666 3356 3786), and the **origin AS** (3786).

Update Example

```
update|A|1499385779.000000|routeviews|route-
views.eqix|None|None|11666|206.126.236.24|210.180.224.0/19|206.
126.236.24|11666 3356 3786|11666:1000 3356:3 3356:2003 3356:575
3786:0 3356:22 11666:1002 3356:666 3356:86|None|Nonefil
```

RIBs

Another type of record is a Routing Information Base (RIB) record.

RIB records have the following format:

```
<dump-type>|<dump-pos>|<record-
ts>|<projfilect>|<collector>|<router-name>|<router-ip>|<record-
status>|<dump-time>
```

RIB Example

The following is a Routing Information Base (RIB) record example.

Consecutive | characters indicate fields without data.

```
R|R|1445306400.000000|routeviews|route-
views.sfmix|||32354|206.197.187.5|1.0.0.0/24|206.197.187.5|3235
4 15169|15169|||
```

BGP Attributes

A detailed explanation of BGP attributes can be found here:

https://www.rfc-editor.org/rfc/rfc4271.txt

As previously mentioned, you will not have to interact with records in this form because PyBGPStream presents records in a way that is more accessible with code, but it will be very helpful to understand the above examples.

Required Python Packages

Plotting Packages

For the plotting and report portion of this assignment, you are required to use the `matplotlib` library ONLY. To use `matplotlib` in the class virtual machine you will need to install the `gi-cairo` backend package. This can be done with the following command:

```
sudo apt install python3-gi-cairo
```

Stats Packages

In this project you will create several Empirical Cumulative Distribution Function (**ECDF**) charts using the statsmodels package:

https://www.statsmodels.org/stable/generated/statsmodels.distributions.empirical_distribution.ECDF.html

For this portion, you are **required to use the statsmodels package only**. To use statsmodels you will need to install the Python package in the VM. This can be done with the following command:

```
pip3 install statsmodels
```

Then in your python code you can call:

```
from statsmodels.distributions.empirical_distribution import ECDF
```

More information on what an **ECDF** is and how to use it can be found at these links:
https://machinelearningmastery.com/empirical-distribution-function-in-python/
https://towardsdatascience.com/what-why-and-how-to-read-empirical-cdf-123e2b922480

Task 1. Historic Measurements - Prefixes and AS Growth Over Time

In this task you will look at how the number of advertised prefixes and ASes has grown over time. The growth of unique prefixes contributes to ever-growing routing tables handled by routers in the Internet core. For students who would like to know more about this topic you may *optionally* read this article:

https://labs.ripe.net/author/vastur/the-shape-of-a-bgp-update/

Using Cached Data

Locate the directory `rib_files` included with this assignment. This directory contains the 8am RIB dump for the first day of January for each year from 2013 to 2021 (inclusive).

Each file contains a timestamp. For example, the file named:

```
ris.rrc06.ribs.1357027200.120.cache
```

includes the collector name (RIS RRC06) the type of data (ribs) and the Unix timestamp of the data (1357027200 which is January 1, 2013 8:00:00 AM).

These cache files serve as a snapshot of BGPM data collected by the collector at the time of the timestamp. In the rest of this assignment the term "snapshot" refers to the data in a particular cache file.

Read / Process / Filter

You will need to write code to process the data for all the cache files in the `rib_files` directory with PyBGPstream. A good idea is to base your code on the paper and examples provided earlier in this document.

Your code will need to set the stream data_interface to singlefile with:

```
pybgpstream.BGPStream(data_interface="singlefile")
```

After `data_interface` has been set make sure the interface options are set with:

```
stream.set_data_interface_option("singlefile", "rib-file",
                                 filepath)
```

where `filepath` will need to be replaced with the path to a specific cache file.

Task 1 – Part A

Using the data from the cache files you will plot a **line graph** that shows the number of total **unique IP prefixes over time**. Towards this goal, complete the function `calculateUniqueIPAddresses` in the `bgpm.py` file. Then modify `bgpm.py` to include matplotlib and write a function that uses output from `calculateUniqueIPAddresses` to create the line graph. This function can have a name of your choice. In your graph:

1. The X-axis should represent time (e.g., January 2013, January 2014 and so on)

2. The Y-axis should represent the number of unique prefixes within the specific time period (e.g. January 2013, January 2014 and so on)

Task 1 – Part B

Using the data from the cache files plot a **line graph** that shows the number of total **unique ASes over time**. Towards this goal, complete the function `calculateUniqueAses` in the `bgpm.py` file **.**

Then write a function that uses output from `calculateUniqueAses` to create the line graph. This function can have a name of your choice. In your graph:

1. The X-axis should represent time (e.g., January 2013, January 2014 and so on).
2. The Y-axis should represent the number of unique ASes within the specific time period (e.g. January 2013, January 2014 and so on).

Task 1 – Part C

For each origin AS, examine the growth of the total unique prefixes they advertise over time. To achieve this:

1. Identify the first and the last snapshot where the AS appeared in the dataset.
2. Calculate the percentage increase of the advertised prefixes, using the first and the last snapshot
3. Report the top 10 origin ASes according to this metric. If you identify more than 10 ASes, with the same growth randomly pick 10 from your set.

Towards this goal, complete the function `examinePrefixes` in the `bgpm.py` file so that it performs the processing and reporting of the ASes.

For Task 1 Parts A,B,C include the graphs and all written answers in your report.pdf file

Task 1 – Part D (Optional and Not for Credit Question)

What do you notice about the trend of overall growth of prefixes, overall growth of ASes? This question is asking for trends. Explain your logic and approach. One example approach is to check the number of prefixes that ASes have for the very last cache file. From there you can handpick a couple that are the top, mean, and bottom of that distribution and track those over time to get an idea of the general trend.

Task 2. Historic Measurements - AS-Path Length Evolution Over Time

In this task you will explore if ASes are reachable over longer or shorter path lengths as time progresses. Towards this goal you will investigate the AS path lengths, and how they evolve over time.

Using Cached Data

For this task use the same data as in Task 1.

Task 2 – Part A

Calculating AS-Path Length

For each snapshot (e.g. Jan 2014) you will compute the shortest AS path length for each origin AS. Specifically, given a snapshot of one month (e.g. Jan 2014), follow the steps below:

- Locate the origin ASes. For example for the path: `11666 3356 3786` AS 3786 is the origin AS.
- For each origin AS (referred to here as "X") collect all the paths for which X appears as the origin AS. Compute the length of each path by considering each AS only once. In other words you want to remove the duplicate entries for the same AS in the same path, and count the total number of unique ASes in the path.
- Among all the paths for X within the snapshot, compute the shortest path length.
- Corner cases: If an AS path has a single unique AS then the path length is 1. If an AS path contains sets of ASes in brackets, e.g.: AS-X AS-Y AS-Z {AS-Y AS-G} AS-C AS-V then count the set {AS-Y AS-G} as *one unique AS* for the purpose of calculating the AS path length,

even though AS-Y is repeated outside the brackets. In the example above, AS-Y and {AS-Y AS-G} count as two ASes. (This is related to aggregation which you can read more about here: https://www.rfc-editor.org/rfc/rfc4271.txt)

In the `bgpm.py` file, complete the function `calculateShortestPath`.
See the function docstring for details.

Task 2 – Part B

For each snapshot, put together all the shortest path lengths for all origin ASes and plot an Empirical Cumulative Distribution Function (ECDF) of those lengths. Therefore, you should have a separate ECDF (graphed with a solid line) for each snapshot (e.g., January 2013, January 2014 and so on). Your function should use the output from the previously written `calculateShortestPath` function.

Finally, plot all ECDFs on one figure and provide a label for each ECDF (and the axes) on your plot and include this in the report. Please see again the required packages section in the beginning of this document, about which python package you are required to use, and example code.

Include your graph in your report.pdf file.

Task 2 – Part C (Optional and Not for Credit Question)

1. From the shortest paths you have already computed, locate the top 10 origin ASes with the longest paths per month.
2. Do those ASes remain the same as time progresses?
3. What is the trend of the AS path lengths progressing over time?

Task 3. Blackholing Events

In this task you will investigate Blackholing events.

Required Background: First you will need to become familiar with Blackholing events. Towards this goal, please look through the following resources:

1. Blackhole Event Description

https://tools.ietf.org/html/rfc7999#section-2

2. Example Blackhole Event

https://blog.apnic.net/2019/03/22/bgp-communities-a-weapon-for-the-internet-part-2/

2. Watch an Example Demo

https://www.youtube.com/watch?app=desktop&v=ot4QUqgunBc

Using Cached Data

Locate the directory `update_files_blackholing` included with this assignment. This is a collection of BGP Updates from January 2, 2021 5:00 to January 2, 2021 9:00. As in previous tasks you will write code to process all the update cache files.

For update files your code will need to set the stream data_interface option to with:

```
stream.set_data_interface_option("singlefile", "upd-file",
                                 filepath)
```

where `filepath` will need to be replaced with the path to a specific cache file.

Task 3 – Part A

Identifying Blackholing Events

Identify events where the IPV4 prefixes are tagged with a Remote Triggered Blackholing (RTBH) community and measure the time duration of the RTBH events. The duration of an RTBH event for a prefix and for a given peerIP, is the time elapsed between the last Announcement that is

tagged with an RTBH community value and the first Withdrawal of the IPV4 prefix. In other words, we are looking at the stream of Announcements and Withdrawals for a given prefix and for a given peerIP, and we are only looking for an **explicit** Withdrawal for an RTBH tagged prefix.

Calculating Event Duration

For a single prefix and for a single peerIP consider the stream:

A1 A2 A3_RTBH_tagged A4_RTBH_tagged W1 W2 W3 W4

- Compute the duration as the time difference between A4_RTBH_tagged and W1.
- Note that in a stream you can have more than one RTBH events. For example, in the following stream A1 A2 A3_RTBH_tagged A4_RTBH_tagged W1 W2 W3 W4 A5_RTBH_tagged W5 we have two events: A4_RTBH_tagged W1 *and* A5_RTBH_tagged W5.
- In case of duplicate announcements, use the latest.
- Consider only non-zero duration events.
- In order to identify RTBH events, look through the AsWs-streams of all peerIPs. (As we mentioned above, just process each AsWs-stream for one peerIP at a time.)

In the `bgpm.py` file, complete the function `calculateRTBHDurations`.

See the function docstring for details.

Task 3 – Part B

Using the data from the cache files plot an ECDF of all the event durations.

This function should use the output from `calculateRTBHDurations` written in the previous step. Make sure the graph and its axes are clearly labeled and include the graph in your report.

Task 3 – Part C

1. How many unique prefixes are associated with RTBH events?

2. What is the total number of RTBH events you identified (for all your prefixes)?

Task 3 Part B,C include the graphs and all written answers your report.pdf file

Task 4. Durations of Announcements Withdrawal Events

In this task, we will measure how long prefix Announcements last before they are getting withdrawn. In this task, we will ***only consider explicit*** Announcement Withdrawals (AW) - not implicit. An explicit AW withdrawal occurs when a prefix is advertised with an Announcement, and then it is withdrawn with a W. (In contrast, an implicit withdrawal occurs with a prefix is announced with an Announcement and then another Announcement follows with different BGP attributes).

Using Cached Data

Locate the directory `update_files` included with this assignment. This is a collection of BGP Updates for a two-hour window starting at 21-01-02 05:10. As with previous tasks, you will need to write code to process the cache files.

Calculating Event Duration

To compute the duration of an Explicit AW event, for a given prefix, you will need to monitor the stream of Announcements (As) and Withdrawals (Ws) separately per peerIP.

- For example, for a prefix, and for a peerIP you observe the following stream: A1 A2 A3 W1 W2 W3 W4 the duration of the explicit AW event for this prefix is the time difference between A3 and W1.
- In this example stream A1 A2 A3 W1 W2 W3 W4 A4 A5 W4 we have two AW events W1-A3 and W4-A5.
- We consider only non-zero AW durations.

Task 4 – Part A

In the `bgpm.py` file, complete the function `calculateAWDurations`. For each prefix, the function should calculate the durations of all explicit AW events it is associated with. Read the function docstring for details.

Task 4 – Part B (Optional Not for Credit Questions)

1. Which are the tuples <advertised prefix IPV4 – peerIP> with the top 10 shortest AW durations?
2. What are the corresponding durations?

Submission

To submit this project, zip your `bgpm.py` and `report.pdf` files and submit to Canvas. The zip file should follow the usual convention:

`gtlogin_bgpm.zip`

Do not include any other files or directories with your zipped submission.

Grading Rubric

Points	Task to be completed
10	Task 1 - Part A
10	Task 1 - Part B
10	Task 1 - Part C
20	Task 2 - Part A
10	Task 2 - Part B
20	Task 3 - Part A
5	Task 3 - Part B
5	Task 3 - Part C
10	Task 4 – Part A
100	**Total Points**

What You Are Allowed to Share

1. Do not share any code.
2. **Watermarked** plots can be shared on Edstem.

Honor Code / Academic Integrity / Plagiarism

Please refer to the Georgia Tech Honor Code located here:

https://policylibrary.gatech.edu/student-affairs/academic-honor-code

We strictly enforce Section 3. Student Responsibilities including these prohibited actions:

- Unauthorized Access: Possessing, using, or exchanging improperly acquired written or verbal information in the preparation of a problem set, laboratory report, essay, examination, or other academic assignment.
- Unauthorized Collaboration: Unauthorized interaction with another Student or Students in the fulfillment of academic requirements.
- Plagiarism: Submission of material that is wholly or substantially identical to that created or published by another person or persons, without adequate credit notations indicating the authorship.
- False Claims of Performance: False claims for work that has been submitted by a Student.