BGP Measurements Project

Table of Contents

Motivation

In this assignment, we will explore Internet Measurements, a field of Computer Networks which focuses on large scale data collection systems and techniques that provide us with valuable insights and help us understand (and troubleshoot) how the Internet works. There are multiple systems and techniques that focus on DNS measurements, BGP measurements, topology measurements, etc. There are multiple conferences in this area, which we invite you to explore and keep up with the papers that are published. For example, the IMC conference is one of the flagship conferences in this area: ACM Internet Measurement Conference

A gentle introduction into the Internet Measurement field is to work with large scale BGP measurements and data to study topics such as: Internet growth over the last 10 years (for example prefixes, Autonomous Systems (AS)), infer problems related to short-lived Announcement and Withdrawals, infer possible DDoS attacks under the way by identifying deployed BGP based countermeasures like “Remote Triggered Blackholing”.

Introduction

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.

Folder Structure and Canvas Submission

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
check_solution.py
```

This project only has a coding component. You will need to complete the functions in the `bgpm.py`. The file `bgpm.py` is the only file that you need to modify.

Before submitting your assignment, make sure all your code works with this folder structure on the course VM. Your code submission will be auto graded in the course VM.

The included `check_solution.py` file includes a basic set of tests for you to run to make sure the code in your bgpm.py file is formatted properly. Passing all these tests may still not guarantee a full grade.

This project is designed to work in the class VM where the BGPStream libraries are installed. Your code will need to run without modification in 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 resources

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://dl.acm.org/doi/pdf/10.1145/2987443.2987482

This paper provides useful background and practical examples of BGPStream so be sure you read it. Additionally, a practical illustrates about how the BGP collection system works is here:

https://afrinic.net/blog/327-on-the-african-peering-connectivity-revealable-via-bgp-route-collectors

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 run the 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 BGP Record Format and BGP Attributes

Moving forward with the assignment, it is useful to familiarize yourself with the BGP record format, the fields (attributes), and their meaning. A detailed explanation of BGP records and attributes can be found here:

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

Also, the BGPReader command-line tool provides access to BGP records and their attributes, which you can use for illustration purposes.

In this assignment you ***will not*** have to interact with the BGP records in this format, because PyBGPStream (the main tool you will be working with) allows you to extract the BGP attributes from BGP records using code. Nevertheless, it is helpful to look through some examples and understand the fields.

Here, we will show sample command line output from BGPReader for ***illustration*** purposes.

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|None
```

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|||
```

Allowed Python Packages

Stats Packages

In this project you have the option of creating several Empirical Cumulative Distribution Function (**ECDF**) charts using the statsmodels package:

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

In a separate Python file for graphing purposes only, 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
https://www.youtube.com/watch?v=_2VDsKU-7SU

No further imports are allowed. All allowed imports are already installed in the class VM.

Tasks

Task 1. Understanding the BGP Routing table Growth: Growth of Prefixes and AS Growth Over Time

In this task you will measure the AS and advertised prefixes growth over time. The growth of unique prefixes contributes to ever-growing routing tables handled by routers in the Internet core. As optional background reading, please read the seminal paper by Towsley et al. On Characterizing BGP Routing Table Growth

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.7057&rep=rep1&type=pdf

Using Cached Data

Locate the directory `rib_files` included with this assignment. This directory contains RIB dump files. Each file contains a timestamp. For example, the file named:

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

includes the collector's 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.

Do not pull your own data. Your solution will be graded on the cached data only.

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 cache files, measure the number of **unique advertised prefixes over time**. **Filter for IPV4 only**. **Also, for the entire assignment, we consider IPV4 prefixes only**. Sort the cache files first from oldest to newest. For each Month-Year measure the number of unique prefixes within that specified time period (Jan-2013, Jan-2014, Jan-2015, … etc).

Towards this goal, complete the function `calculateUniquePrefixes` in the `bgpm.py` file. Make sure that your function returns the measured volumes over time in the required data structure as explicitly explained in `bgpm.py`.

Task 1 – Part B

Using the data from the cache files, measure the number of unique **AS over time**. Just as before sort the cache files first from oldest to newest. **Filter for IPV4 only**. For each Month-Year measure the number of unique **ASes** within that specific time period (Jan-2013, Jan-2014, Jan-2015, ... etc.) considering all paths in the specific time period. Here, we consider all AS that appear in the paths (not only the origin AS). You may encounter corner cases of paths with the following form: AS-X AS-Y AS-Z {AS-Y AS-G}. In this case, consider the AS in the brackets as a single AS. So, in this example, you will count 4 distinct AS.

Towards completing this measurement, edit the function `calculateUniqueAses` in the `bgpm.py` file. Make sure that your function returns the measured volumes over time in the required data structure as explicitly explained in `bgpm.py`.

Task 1 – Part C

Again, use the same cache files and sort them first from older to newer. Filter for IPV4 only. Consider each origin AS separately and measure the growth of the total unique prefixes advertised by that AS over time. To compute this, for each origin AS:

1. Identify the first and the last snapshot where the origin AS appeared in the dataset.
2. Calculate the percentage increase of the advertised prefixes, using the first and the last snapshots.

3. Report the top 10 origin AS according to this metric.

Towards this goal, complete the function `examinePrefixes` in the `bgpm.py` file for your code to perform the calculations, and reporting of the AS.

Corner case: When calculating the prefixes originating by an origin AS, you may encounter paths of the following form: AS-X AS-Y AS-Z {AS-Y AS-G}. This is a corner case, and it should affect only a small number of prefixes. In this case, you consider the entire set of AS {AS-Y AS-G} as the origin AS.

Task 1 – Part D Optional Ungraded Questions

In this section, you have the option to work separately on additional questions. Do not submit your solution/code for these optional questions with the required tasks of this assignment.

1. Represent your measurements as graphs. Using Matplotlib, plot a **line graph** that shows the number of total **unique IP prefixes over time**. In your graph. The X-axis should represent time (e.g., January 2013, January 2014 and so on). The Y-axis should represent the number of unique prefixes within the specific time period (e.g. January 2013, January 2014 and so on). Similarly, repeat for the growth of unique AS over time.
2. What do you notice about the trend of overall growth of prefixes, and overall growth of AS? What is your hypothesis about the factors that contribute the most to the growth? Explain your logic and approach.

Task 2. Routing Table Growth: AS-Path Length Evolution Over Time

In this task you will measure if an AS is reachable over longer or shorter path lengths as time progresses. Towards this goal you will measure the AS path lengths, and how they evolve over time. **Again, we consider IPV4 prefixes only**. We use the same cached 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 that is present in the snapshot. Specifically, given a snapshot of one month (e.g. Jan 2014), follow the steps below:

- Locate the origin AS that are present in the snapshot. For example, for the path: |11666 3356 3786| the AS 3786 is considered as the origin AS.
- For each origin AS collect all the paths for which it appears as the origin AS.
- Compute the length of each path by considering each AS in the path 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 AS in the path.
- ***Example:*** Consider the path |AS-D AS-C AS-B AS-C AS-A|. The AS-A is the origin AS. AS-C appears twice in the path. This is a path of length 4.
- Among all the paths for an AS within the snapshot, compute the shortest path length.
- Filter out all paths of length 1.
- ***Corner cases:*** The data that we are working with are real data, which means that there may be ***few*** corner cases. In the vast majority of the cases, paths have a straightforward form of |AS-D AS-C AS-B|. Still there are corner cases as follows:
    a. If an AS path has a single unique AS then the path length is 1. As mentioned above, ignore all paths of length 1. Or if a path is AS-A AS-A AS-A again it has a length of 1 and it is ignored.
    b. If an AS path contains sets of multiple AS in brackets. This is related to aggregation which you can read more about here: https://www.rfc-editor.org/rfc/rfc4271.txt). ***Example***: |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 different ASes. The path length in this case is 6. Similarly, in this case |AS-X AS-W AS-Y AS-Y {AS-Y}| the path length as 4. AS-Y is duplicate outside the brackets, and {AS-Y} counts as one unique AS.

c. You can ignore all other corner cases.

For each snapshot separately, put together all the shortest path lengths for all origin AS that appear in that snapshot. So, if an origin AS is present in Jan-2014, but not it is not present in a Jan-2015 snapshot, then only include that origin AS in the Jan-2014 snapshot.

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

Task 2 – Part B Optional Ungraded Questions

In this section, you have the option to work separately on additional questions. Do not submit your solution/code for these optional questions with the required tasks of this assignment.

1. For each snapshot separately, put together all the shortest path lengths for all origin AS that appear in that snapshot. So, if an AS is not present in a snapshot, then do not include that in the snapshot statistics that you are calculating. Using this measured data run the Empirical Cumulative Distribution Function (ECDF) for the lengths of each snapshot. Therefore, you should have a separate ECDF for each snapshot (e.g., January 2013, January 2014 and so on). Your function should use the output from the previously written `calculateShortestPath` function.
2. Plot all ECDFs on one figure and provide a label for each ECDF (and the axes) on your plot.
3. What is the trend of the AS path lengths progressing over time?
4. From the shortest paths you have already computed, locate the top 10 origin AS with the longest paths per month.
5. Do those AS remain the same as time progresses?

Task 3. Blackholing Events

In this task you will identify and measure 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/

3. 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.

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.

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** of the IPV4 prefix 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 IPV4 refix 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 event. 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 this example stream, A1 A2 A3_RTBH A4 A5 W1 W2, there is no explicit withdrawal event. The announcement A3_RTBH followed by A4 is an implicit withdrawal.
- In case of duplicate announcements, use the latest.
- Consider only non-zero duration events.
- 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.)
- As far as the time unit, keep the events durations in seconds, in other words, no transformation is needed, if you simply subtract the timestamps of As and Ws as they are in the data.

In the `bgpm.py` file, complete the function `calculateRTBHDurations`. See the function docstring for details. Your function needs to return the peerIPs, the prefixes and the durations of the RTBH events in the specified data structure.

Task 4. Durations of Announcements Withdrawal Events

In this task, we will measure how long prefix Announcements last before they are getting withdrawn. This matters because when a prefix gets Advertised and then Withdrawn, this information propagates through, and it affects the volume of the associated BGP traffic. Optional background reading on this topic: https://labs.ripe.net/author/vastur/the-shape-of-a-bgp-update/

Here we will measure how long prefix Announcements last before they are withdrawn, and 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). Again, we consider IPV4 prefixes only.

Using Cached Data

Locate the directory `update_files` included with this assignment. As with previous tasks, you will need to write code to process the cache files.

Calculating AW 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. Again, we are only looking for **last A and first W.**
- In the following example stream (associated with a specific prefix and a specific peerIP) A1 A2 A3 W1 W2 W3 W4 A4 A5 W4 we have two AW events W1-A3 and W4-A5.
- In this example stream (associated with a specific prefix and a specific peerIP), A1 A2 A3 A4 A5 W1 W2, there is one explicit withdrawal event W1-A5. The announcement A3 followed by A4 is an implicit withdrawal.
- We consider only non-zero AW durations.

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. Your function needs to return the peerIP, the prefixes and the durations of the AW events in the specified data structure.

Task 4 – Optional Ungraded Questions

In this section, you have the option to work separately on additional questions. Do not submit your solution/code for these optional questions with the required tasks of this assignment.

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, submit `bgpm.py` to Canvas. **Only include bgpm.py in your submission.**

`bgpm.py`

Grading Rubric

Points	Task to be completed
10	Task 1 - Part A
10	Task 1 - Part B
10	Task 1 - Part C
30	Task 2
20	Task 3
20	Task 4
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.