1 Overview

In this programming assignment, you will have the opportunity to implement a sketch-based algorithm designed to monitor network traffic across a high-traffic network link. The primary aim of this assignment is to help you understand the delicate balance between memory usage, computational resources, and the accuracy achieved when employing sketch-based estimation algorithms in real-world network monitoring scenarios. We will provide you with real-world network traces and a skeleton code structure to guide you in completing tasks related to network monitoring.

References: Lecture 3,4 - Traffic Models and Network Monitoring

Honour Code *This assignment constitutes **10%** of your final grade for this module. You are expected to work on the assignment **individually**. While you can discuss with your classmates and refer to the internet, we will not condone plagiarism. You are expected to adhere to the NUS code of conduct.*

2 Submission Instructions

Items to be submitted:

- *Source code* (`count_min.py, test.py`) - This is where you fill your code.
- *Report* (report-student_id.pdf) - Your response report. Please mention your name and Student ID(AXXXXXXXY) on the top left corner of the first page of your response file. The report should briefly explain your algorithmic approach and present the results of your implementations.

Zip the two files together, name it in the following format: **cs5229_StudentID.zip** (e.g., cs5229_A1234567X.zip), and submit to Canvas folder *"Programming Assignment 1"*.

Submission Deadline: 1 October 2023, 23:59pm Late submissions will be penalized with 20% of the total grade for each day of late submission.

3 Network Monitoring

Network measurement is of utmost importance for ensuring the efficient operation of networks. To facilitate effective network management, it becomes essential to obtain fine-grained measurements, including metrics such as per-flow size, heavy hitter detection, and cardinality, among others. These measurements play a pivotal role in various network management tasks, encompassing load balancing, congestion control, quality of service assurance, scheduling optimization, and anomaly detection. Software switches have emerged as versatile platforms for conducting network measurements. However, a significant challenge persists. Network link speeds have now reached a staggering 400 Gbps, and switching capacity has exceeded tens of Tbps. Consequently, there is a pressing need to address the scalability of

software-based processing speeds to keep pace with the ever-increasing demands of high-speed networks.

One approach to network monitoring is to capture and compute aggregated statistics directly in the data plane. These aggregated data can then be further processed in the control plane for more complex tasks. In this assignment, you will gain insights into the utilization of sketches as a means of aggregating statistics in the data plane. This approach allows for efficient preliminary data analysis within the network infrastructure itself before forwarding the relevant information to the control plane for more advanced processing and decision-making.

The Need for Sketch-Based Techniques:

Let's examine the example of cardinality estimation, where the primary goal is to determine the number of unique flows within a network trace.

- While a naïve approach, such as a *counter-based* or *set membership-based* method, might suffice for a small number of unique flows in the network, these solutions become impractical in real-world scenarios. The primary challenge arises from the fact that the memory requirements for these solutions increase linearly with the scale of data. For instance, the utilization of counters for each flow would demand several terabytes of storage, far surpassing the capabilities of typical commodity hardware and making them unsuitable for practical implementation.

- An alternative consideration could be to capture and process packets *offline*. However, given the high network link speeds, the volume of data collected would be massive. Moreover, several critical network monitoring tasks necessitate real-time monitoring of the network to ensure its efficient operation.

- Another approach might involve *sampling-based* techniques, where a subset of the entire network trace is considered to estimate network metrics, thus reducing the measurement overhead. However, it's important to note that this method cannot provide highly accurate and fine-grained statistics. Often, there is an inevitable loss in measurement resolution and accuracy when employing sampling-based techniques.

In light of these challenges, sketch-based techniques emerge as a promising solution for addressing the complexities of network monitoring, offering a way to achieve accurate measurements and analysis while efficiently managing memory usage and computational resources.

The Count-Min Sketch

In the lecture, you've been introduced to essential concepts such as the Bloom Filter [2, 1] and sketches, with a particular emphasis on the Count-Min sketch [4]. These techniques, often referred to as *'lossy compression'* methods, offer the advantage of representing network state in sub-linear space. The underlying insight driving the adoption of these techniques is that, in practical applications, an *exact* answer isn't always necessary. Instead, what matters is obtaining a *reasonably accurate* estimate.

The Count-Min sketch, in particular, has found its way into real-world systems. For instance, at Google, a precursor of the Count-Min sketch, known as the "count sketch", has been implemented on their MapReduce parallel processing infrastructure. Overall, the Count-Min Sketch has since found application in a wide array of network analyses [3].

In this assignment, our primary focus will be on the Count-Min sketch. We encourage you to explore its various variants and potential improvements as you work toward achieving the objectives outlined in the assignment. This will provide you with hands-on experience in leveraging sketch-based algorithms for effective network monitoring and measurement.

We recommend that you refer to resources, such as this paper [4], to explore error analysis and design choices when implementing a Count-Min Sketch.

4 Setup

Please obtain the required files from `ProgrammingAssignment1.zip`. You can install the necessary Python modules using the following command:

```
pip install -r requirements.txt
```

You are **not** allowed to import any additional libraries for your implementation.

4.1 Input Data

Live internet traffic is captured as a `.pcap` file, which contains packet headers and payloads. We extract the essential features from the `.pcap` file into a `.csv` file. You can unzip the packet trace(`packet-trace.csv`) in the `Data/` folder. This data includes the following information:

- **Packet Header Fields (X)**: These fields contain information about the packet's header, which is used to identify and categorize the 5-tuple flow. They include: `Source_IP, Destination_IP, Source_Port, Destination_Port,` and `Protocol`.
- **Packet Length (`frame.len`)**: This field specifies the frame length of the packet.

All packets sharing the same header fields X are considered part of the same **flow**.

5 Assignment Objectives

In this assignment, your objective is to design and implement a data structure based on the Count-Min Sketch algorithm that can simultaneously achieve the goals outlined in the following sections. You will implement this sketch and its functionalities within the `CountMinSketch` class, which resides in the `count_min.py` file.

5.1 Task I - Flow Frequency Estimation [15 points]

Your implementation should be capable of counting the number of times a specific flow appears in the given data trace. You are required to complete the `estimate_frequency(self, flow_X)` function to achieve this task.
Input: Packet flow `(flow_X)` being queried.
Output: An integer value indicating the observed frequency for the given flow.

5.2 Task II - Cardinality Estimation [15 points]

Cardinality refers to the number of unique flows observed in the given network trace. You need to complete the `count_unique_flows()` function to implement this task.
Output: An integer value indicating the observed number of unique flows.

5.3 Task III - Heavy Hitter Detection [10 points]

In this assignment, a "Heavy Hitter" refers to one of the top 100 observed flows in terms of flow size. You are required to complete the `find_heavy_hitters()` function to accomplish this task, and report all the heavy hitters in the given packet trace.
Output: 5-tuples representing the heavy hitter flows and their corresponding flow sizes.

Important Note: Your assignment involves designing and implementing a Count-Min Sketch variant capable of handling all three tasks - Flow Frequency Estimation, Cardinality Estimation, and Heavy Hitter Detection - using a single instance of the data structure. In other words, your goal is to create a versatile Count-Min Sketch-based solution that can address various network queries without necessitating the creation of a new data structure for each task. Simultaneously, you should strive for *efficient memory utilization*. This approach mirrors real-world scenarios where a single sketch serves multiple network monitoring needs, emphasizing the practicality of your implementation.

5.4 Initializing the Sketch

To enable the functionalities discussed above, you must complete the following:

1. `__init__(self)`: In this function, your task is to initialize the sketch according to your preferred design. Use the `self.sketch` attribute for storing your sketch. Additionally, optional auxiliary storage can be utilized and stored in the `self.auxiliary_storage` attribute. Please ensure that the total memory allocated for both the sketch and auxiliary storage remains below 1 Megabyte. You are also allowed to define your own initialization parameters for this function. Make sure to adjust the class object initialization in the `test.py` file accordingly.

2. `hash_func(self, packet_flow, i=None)`: Implement this function to return the hash values corresponding to a given flow. The optional argument `i` can be used to specify the hash function ID, which is particularly useful in scenarios involving multiple hash functions.

3. `add_item(self, packet_flow, packet_len)`: Use this function to update the sketch for each packet as they stream through the network. The function accepts the `packet_flow` representing the 5-tuple header fields and the packet length `packet_len`.

6 Grading Policy

Your code will undergo assessment using unseen packet traces based on the following criteria:

- Accuracy of estimations in all three tasks.
- Memory size of the sketch.
- Quality of code documentation and explanation.

Constraints on Memory Utilization:

To optimize memory usage, there is a specific upper limit imposed on the size of the network sketch in your implementations. We use `Pympler` to monitor the memory usage of the `CountMinSketch` class. While you are allowed to define variables locally within functions, please be aware that you are **not** expected to declare any additional large data structures for processing.

Constraints on Time:

Although we do not impose runtime restrictions, we do expect your code to complete tasks within a reasonable timeframe. Implementations requiring excessively long runtimes may incur penalties.

7 Getting Help

If you encounter any issues or have questions related to the assignment, please raise them on `Piazza`. Ensure that you *tag* your posts under the topic `programming1`. If you are unable to access Piazza, feel free to reach out to the teaching team via email at kanav.sabharwal@u.nus.edu.

References

[1] Bloom Filters. `https://en.wikipedia.org/wiki/Bloom_filter`.

[2] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[4] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.