Group: Meet Desai, Denna Loh

Query-Based Network Monitoring Tool

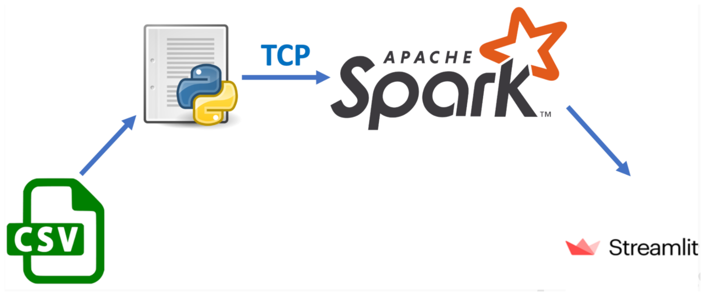
1. Problem Statement

Network monitoring helps you point out the exact location of the network problem or prove that the network is not the problem. Continuous monitoring can help you to identify potential issues before they occur. It means that you can proactively solve problems before they get to the users. It is also essential to identify security threats. Hence, pinpointing and identifying devices that are resource intensive and the degree of utilization as compared to other devices is vital in ensuring that the network continues to function as expected. This could also prevent a skewed allocation of resources or a bandwidth hog from occurring in the first place preventing any effect this could have on the other devices on the network as well. Therefore, a query-based network monitoring tool can be used to extract a utilization profile of a network's resources.

2. Implementation

System Architecture

The components of the system are a csv file containing the data input, a python script, a spark script and a GUI developed using Streamlit.



Data Input

The data source chosen to be used is from a unified host and network data set, which contains 90 days of network event data. Due to the size of the dataset, only day 1 was utilized (Total number of rows = 115,949,436). The data for day 1 was converted and stored in a csv file. The following table contains the fields and the respective description of each field of the data.

Field Name	Description
Time	The start time of the event in epoch time format
Duration	The duration of the event in seconds.
SrcDevice	The device that likely initiated the event.
DstDevice	The receiving device.
Protocol	The protocol number.
SrcPort	The port used by the SrcDevice.
DstPort	The port used by the DstDevice.
SrcPackets	The number of packets the SrcDevice sent during the event.
DstPackets	The number of packets the DstDevice sent during the event.
SrcBytes	The number of bytes the SrcDevice sent during the event.
DstBytes	The number of bytes the DstDevice sent during the event.

Data Streaming

A python script (i.e., “input.py”) was created to stream the data in from the csv file and send to the pyspark script via TCP row by row. The pyspark script then parses each message sent and appends the data to a list.

Spark Implementation

In order to provide a live and continuous monitoring of a network’s resources, the queries of the following type were chosen to be answered:

1. List the devices that are consuming more than H percent of the total external bandwidth over the last T time units.
2. List the top-k most resource intensive devices over the last T time units.
3. List all devices that are consuming more than X times the standard deviation of the average traffic consumption of all devices over the last T time units.

The last T time units were obtained by using sliding windows and T was the size of the window (i.e., windowed data). The slide chosen to be used was 10 units.
Additionally, the queries were also run on all data streamed in since the beginning (i.e., dynamic data).
Spark SQL functions were used to obtain the results of the queries.

To implement a sliding window, the tuples are appended to a list until the list size reaches the required window size. The earliest 10 tuples are then removed and new tuples are appended again.

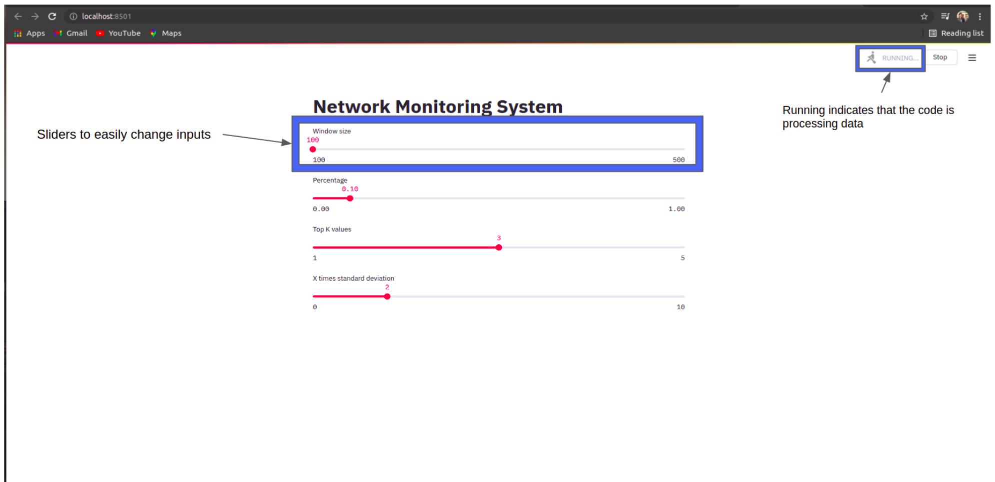
For the dynamic data, the tuples are continuously appended to a list. In other words, the size of the list will only continue to increase until all the data from the csv has been streamed in.

Both lists are then converted into a dataframe and a temporary dataframe containing the sum of total packets from distinct source devices and the respective source device names was created. The data in the temporary dataframe was then used to answer the queries.

Data Visualization

The data was visualised using bar graphs for all 3 queries. The bar graph had the name of the SrcDevice on the x-axis while it had the count of packets on the y axis. The data was visualised using the streamlit library in python, an open source app framework. Streamlit was used to make the GUI. The user input can be taken using 4 different sliders for Window Size, Percentage, Top K values and the X times of standard deviation. Once, the input is given the code performs different queries on the data and then prints out different queries results in the form of a bar graph.

3. Snapshots from Demo



localhost:8501
Apps
Gmail
YouTube
Maps
Reading list
RUNNING...
Stop
Sliders to easily change inputs
Running indicates that the code is processing data
Made with Streamlit

Network Monitoring System

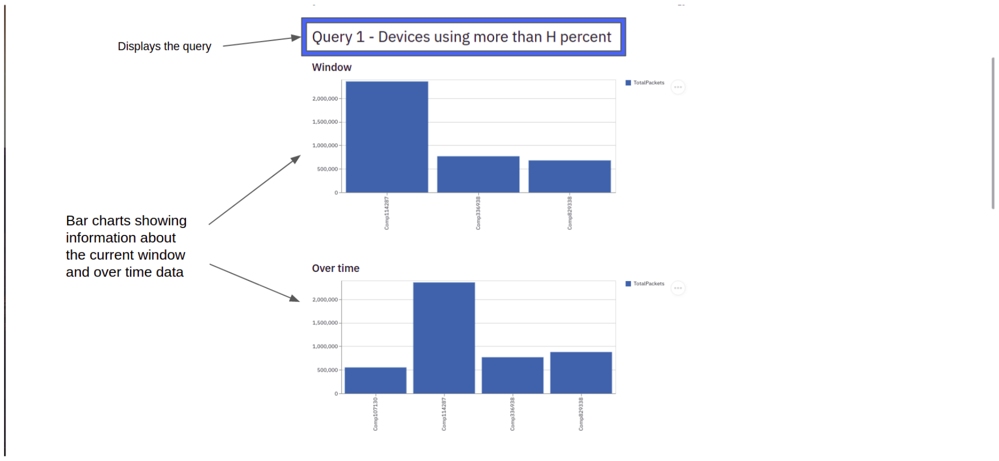
Window size: 100 (range 100 – 500)

Percentage: 0.10 (range 0.00 – 1.00)

Top K values: 3 (range 1 – 5)

X times standard deviation: 2 (range 0 – 10)

Start up Page



Query 1 - Devices using more than H percent

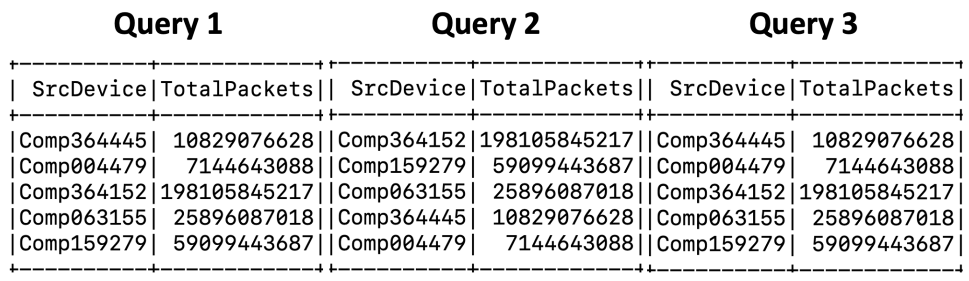
Displaying Queries

4. Streaming Optimization

A streaming optimization that could have been utilised is load shedding. This can be implemented by dropping a certain percentage of packets being streamed in.

5. Streaming Algorithms for Dynamic Data

Storing all the data streaming in from the start may create memory issues. Hence, the utilization of sampling algorithms could reduce the volume of the input data by retaining only a subset for analysis.
The accuracy of the sampling algorithms will be determined by comparing their query results with the query results from a subset of the static data (i.e.100,000 tuples) which can be seen below.

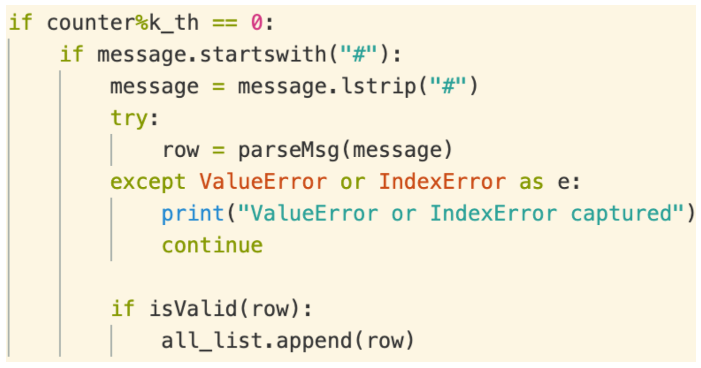


```
        Query 1                    Query 2                    Query 3
+-----------+------------++-----------+------------++-----------+------------+
| SrcDevice|TotalPackets|| SrcDevice|TotalPackets|| SrcDevice|TotalPackets|
+-----------+------------++-----------+------------++-----------+------------+
|Comp364445| 10829076628||Comp364152|198105845217||Comp364445| 10829076628|
|Comp004479|  7144643088||Comp159279| 59099443687||Comp004479|  7144643088|
|Comp364152|198105845217||Comp063155| 25896087018||Comp364152|198105845217|
|Comp063155| 25896087018||Comp364445| 10829076628||Comp063155| 25896087018|
|Comp159279| 59099443687||Comp004479|  7144643088||Comp159279| 59099443687|
+-----------+------------++-----------+------------++-----------+------------+
```

The accuracy value will be the percentage of the number of devices the sample's query results managed to obtain out of the total number of devices (i.e., 15 source devices) from the static data's query results.
Uniform and random sampling algorithms were chosen to be analysed. In both algorithms, a "counter" variable was used to identify the index of the row being streamed in.

Uniform Sampling

If the index of the row streamed in divisible by variable "k_th", it will be parsed and appended to the sample.



```
if counter%k_th == 0:
    if message.startswith("#"):
        message = message.lstrip("#")
        try:
            row = parseMsg(message)
        except ValueError or IndexError as e:
            print("ValueError or IndexError captured")
            continue

        if isValid(row):
            all_list.append(row)
```

Random Sampling (i.e., Reservoir Sampling)

Reservoir sampling was implemented by having the rows continuously appended to a list until the size of the list is equal to the chosen sample size. A random number between 0 and the index of the new row streamed in is then generated. If the number generated is smaller than the sample size, the new row will be appended and will replace the row with the index equal to the random number generated.

```
if len(all_list) < sample_size:
    all_list.append(row)
else:
    index_to_remove = random.randint(0, counter-1)
    if index_to_remove < sample_size:
        all_list[index_to_remove] = row
```

Accuracy

Both algorithms could not yield samples with similar query results for this dataset at all. Even when the variable “k_th” was set to 2 for uniform sampling and the sample size was set to 99,000 (i.e., 99%) for random sampling, only 2 out of 15 of the devices from the static data (i.e., 13.3% accuracy) could be obtained.
Based on the values calculated, in the context of this application, the utilization of sampling algorithms does not allow for yielding accurate results of the queries. This could be due to the numerous distinct source devices in the dataset.

6. Dynamic vs Static Data

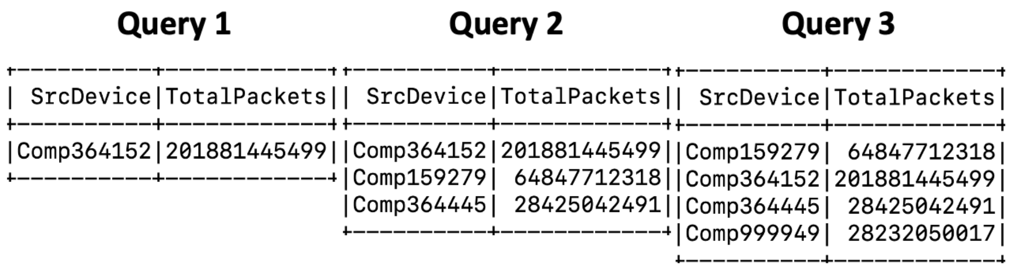
The following queries were used to compare the dynamic and static data (i.e., non-streaming data):

1. Devices that are consuming more than 10% percent of the total external bandwidth in the entire dataset.
2. Top 3 more resource intensive devices in the entire dataset.
3. All devices that are consuming more than 20 times the standard deviation of the average traffic consumption in the entire dataset.

Query Results

It required approximately a total of 25,000,000 rows out of 115,949,436 rows to be streamed in before the query results of the entire dynamic data (i.e., without any sampling algorithm) becomes identical to the query results of the static data. The results are as shown below. However, when utilizing both streaming algorithms, the results never converge to become identical to the query results of the static data.

Dynamic Data



Query 1

```
+----------+------------+
| SrcDevice|TotalPackets|
+----------+------------+
|Comp364152|201881445499|
+----------+------------+
```

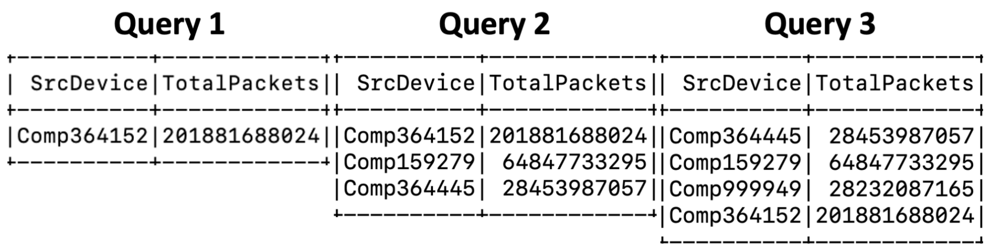
Query 2

```
+----------+------------+
| SrcDevice|TotalPackets|
+----------+------------+
|Comp364152|201881445499|
|Comp159279| 64847712318|
|Comp364445| 28425042491|
+----------+------------+
```

Query 3

```
+----------+------------+
| SrcDevice|TotalPackets|
+----------+------------+
|Comp159279| 64847712318|
|Comp364152|201881445499|
|Comp364445| 28425042491|
|Comp999949| 28232050017|
+----------+------------+
```

Static Data



Query 1

```
+----------+------------+
| SrcDevice|TotalPackets|
+----------+------------+
|Comp364152|201881688024|
+----------+------------+
```

Query 2

```
+----------+------------+
| SrcDevice|TotalPackets|
+----------+------------+
|Comp364152|201881688024|
|Comp159279| 64847733295|
|Comp364445| 28453987057|
+----------+------------+
```

Query 3

```
+----------+------------+
| SrcDevice|TotalPackets|
+----------+------------+
|Comp364445| 28453987057|
|Comp159279| 64847733295|
|Comp999949| 28232087165|
|Comp364152|201881688024|
+----------+------------+
```

7. Future Work

There were a few integers in the dataset that were too large for spark to process as they were greater than 231-1. However, as the percentage of such integers were extremely small as compared to size of the dataset, rows containing integers greater than 231-1 were just dropped. Another challenge faced during implementation of the GUI using streamlit, one major challenge that we faced was that streamlit always reruns the entire code for every single change in the input. This resulted in the code trying to bind with the same socket that it was already using which threw an OSError in python. To circumvent this, We were able to use the st.cache function, which allows us to make a static bind with the socket. Due to streamlit rerunning the code, we still face the issue that everytime a change in the input triggers the code to collect the data from that point onwards, discarding the previous data.

Hence, to improve accuracy, future work could include looking into calculating the sum and average when the integers are split into multiple columns. Also, a solution has to be found that prevents discarding of the previous data by using the st.cache function.

8. References

1. Unified Host and Network Data Set - Cyber Security Research. (2017). [Dataset]. https://csr.lanl.gov/data/2017/
2. Streamlit, open source app framework. https://streamlit.io/