Visualization for Frequent Itemset Mining for Streaming Data

Sophie Qiu

May 16, 2016

1 Introduction

Visualising communication log data is useful in many cases. When large amount of data come in, it is hard to tell what is happening by looking at numbers and logs. Therefore, visualising data makes it easier for human beings to detect abnormal traffic.

There are some techniques presented in the literature on how to visualise network traffics. However, these algorithms are applied on static data rather than streaming data. There are some significant benefits if streaming data can be visualised at real time. It is helpful to detect malicious connection immediately. It also helps human beings to understand the flow of the traffic and how they evolve over time. In this project, I explored how to apply visualisation techniques on streaming data.

2 Related Work

The first work that is useful to my project is by Glatz et al.[1]. In this work, Glatz et al. introduces a schema to visualise network traffic data using hyper graphs. The schema first uses Frequent Itemset Mining (FIM) to identify the most frequent 5 connections. Then the data is visualised as hyper graphs. Each connection is represented by a circle, whose size is proportional to the frequency. Each circle contains the frequency of this connection in the examined period. Then the attributes of these connections, such as the source and destination IPs and Ports, are added to the corresponding circles.

However, their FIM is performed after the traffic data is collected and after the period is ended. Mozafari et al. propose an algorithm, called Sliding Window Incremental Miner(SWIM), using sliding window technique to fast and accurately identify frequent items. Their algorithm divides data into slides and group slides in to windows. Frequencies of items are calculated within each slide. The data of all slides in the same window are then aggregated. For the incoming slide S, mine S and update the frequency on item p if p is found in S. For expiring slide S, update the frequency of p if S has been counted in. This algorithm produces no false positive or false negative with fair speed[2].

3 Method

My work intends to combine these two works and use the graphics package in Python[3] to realise visualisation for streaming data. In this section, I will present each part of the system.

First, streaming data is fed into SWIM to mine the most frequent items within this time frame. The frequency data produced by SWIM is fed into my Python program, which uses *graphics* package to handle visualisation. The way I visualise the most frequent items is inspired by Glatz et al.'s work. They have mentioned two schemas for visualisation, showed in Figure 1 and Figure 2 respectively (figure descriptions are quoted from their paper)[**?**]. In their paper, they point out that plotting traffic data in a parallel-coordinate plot as in Figure 1 is not an ideal schema to visualise large amount of data. However, in the streaming scenario, since we only need to deal with a small set of data at a given moment, this schema actually shows some advantages. I will discuss this in the next section. I also present the data in the schema that they proposed, known as hyper graphs. However, it's application on streaming data is not as ideal as it is on static data and maybe not as straight forward as the first schema.

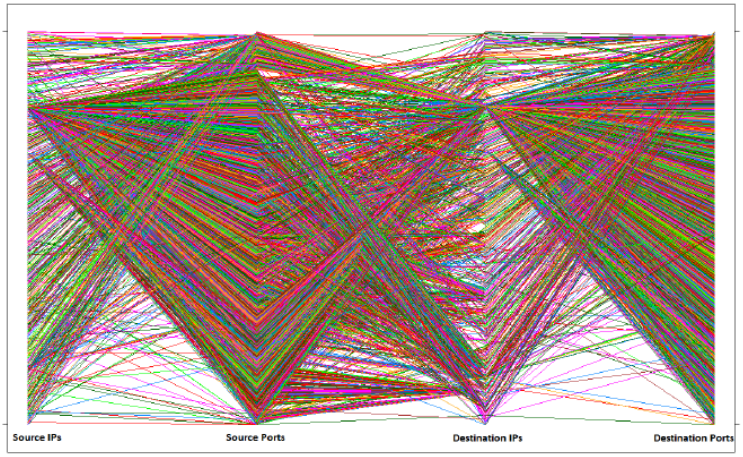

Figure 1: "Visualization of 15,000 traffic flows in a parallel-coordinate plot. The visualization becomes cluttered even with a small number of flow records."

Because of the limit on the storage space of my laptop, the performance of streaming data is emulated. I only took a small part of the data that was given to us. I also did not have enough time to connect all parts of the system. Therefore, data are first fed in SWIM and produce an output file. The Python program I coded then takes the output file and process one window of SWIM data at a time. Thus giving the illusion of incoming data. I think this only requires some engineering work and does not hamper the proof of possibility

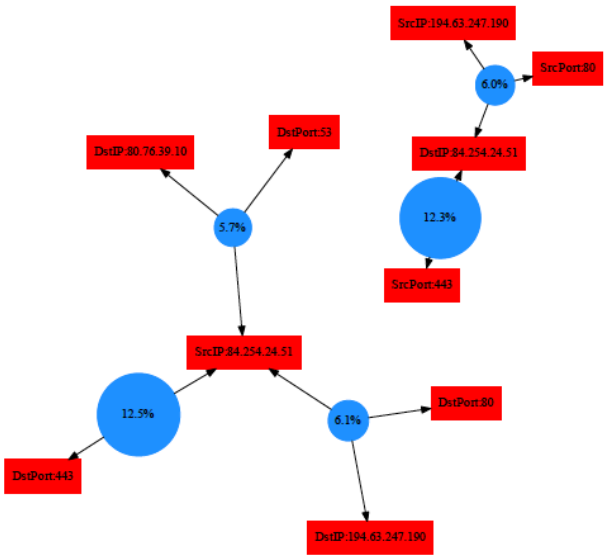

Figure 2: "Frequent itemset visualization of top 5 frequent itemsets extracted from 15,000 traffic flows based on the proposed scheme."

of visualising streaming data, thus did not spent too much time on it.

The SWIM implementation provided by Mozafari accepts only integer value data, because they need to use these value as keys in their trees. It will take a significant amount of work to restructure the code so that i can take in string literals as input. Therefore, I created a hash map that maps source and destination IP address pairs into integer values. i only managed to use IP addresses as input data. It would also be beneficial if source and destination ports are also taken into consideration.

4 Result and Discussion

In this section, I will show 3 types of visualisation schemas that I have implemented and discuss their strength and drawbacks.

4.1 Parallel-coordinate Plot

Figure 3 shows how I incorporates the idea of parallel-coordinate plot schema on streaming data. Each IP address takes one line on either source of destination side. Source and destination IP addresses of each connection is shown by an arrow. The size of red dots is proportional to the frequency of the connection within the current window. Grey circles, text, and arrows are the frequent items from the last window. This can show how data

evolve over time. Currently I only show data from current and previous one window. It is very easy to add more windows to display together.

Because we only have very few items within each window, this schema is cleaner and more strait-forward than that of Glatz et al.'s. According to my observation, some IP addresses appear in early windows will appear again after several windows. Therefore I chose to leave all IP addresses that have appeared so far on the screen. As a result, IP addresses are used as a location indicator. The observer can easily associate one computer with a spot on the screen. However, because it is streaming data, as data comes in, we will eventually run out of space. One possible solution is to delete some IP addresses that have not been seen for a long time in order to save space for incoming data.

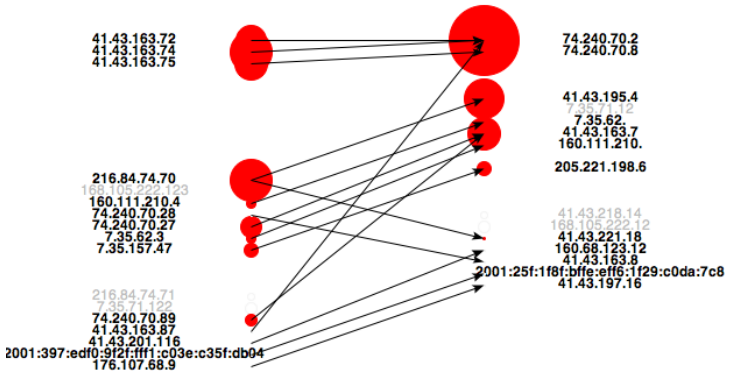

Figure 3: Visualisation of Streaming Data with Parallel-Coordinate Plot

4.2 Hyper Graphs

I cannot infer how Glatz et al. design the positions for the circles, because it is static. They might have chosen it arbitrarily for aesthetic reasons. For simplicity, I assume they never change along the time. Therefore, I pre-select the positions where the circles should reside. The only things left are to adjust the sizes of the circles and print text to indicate source and destination IP addresses.

4.2.1 First Attempt

The first attempt I made for creating hyper graphs is to pre-select a group of positions for circles to be placed. For each window, the most frequent item will take an empty spot if it is a new item and display as bright blue. The circle becomes dark blue if the item appeared

in the last window but disappears in the current window. If the item appears in consecutive windows, the size of the circle changes according to the changes in it's frequency.

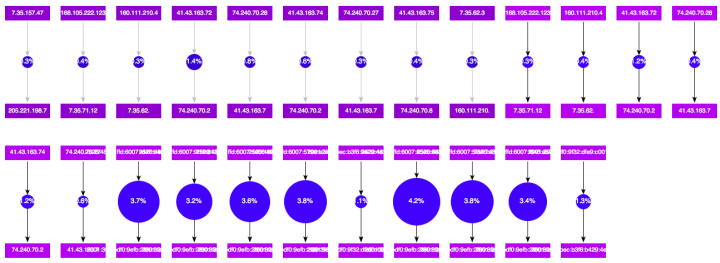

Figure 4: Visualisation of Streaming Data with Hyper Graphs

4.2.2 Second Attempt

The result of the first attempt is ugly because of the arrangement of the circles. I then create a graph that is similar to Figure 2. I set 5 positions for circles and their corresponding text boxes. For each window, I take the items with top 5 frequencies and put them into each circle. The position is improvised based on the tradeoff between my aesthetic standard and my Python drawing skill (the latter significantly hampered the quality of the result and limited my aesthetics expectation on my work). Comparing Figure 5 and Figure 6, which are results of this attempt for relatively lower overall frequencies and higher frequencies, it is desirable if these circles can adjust their positions according to their distances to other circles. This display method fixes the position according to the frequency. The highest frequency always shows up in the middle right circle. However, I have not figured out how to show the evolution of data using this schema.

The implementation can be found on my github: `<https://github.com/sophieball/VisualNW>`

5 Conclusion and Future Work

My implementation explores some schemas to visualise frequent items in streaming network traffic data. Comparing to static data visualisation, there are some significant differences. First, for the static data, we know the number of items so that we can allocate space accordingly, whereas for streaming data, the places and sizes of shapes need to be adjusted constantly in order to create beautiful result and, more importantly, to use the screen wisely. Second, one dimension streaming data provides is the evolution of data over time. It is, however, difficult to display the trace of data in the visualisation. I only explored some

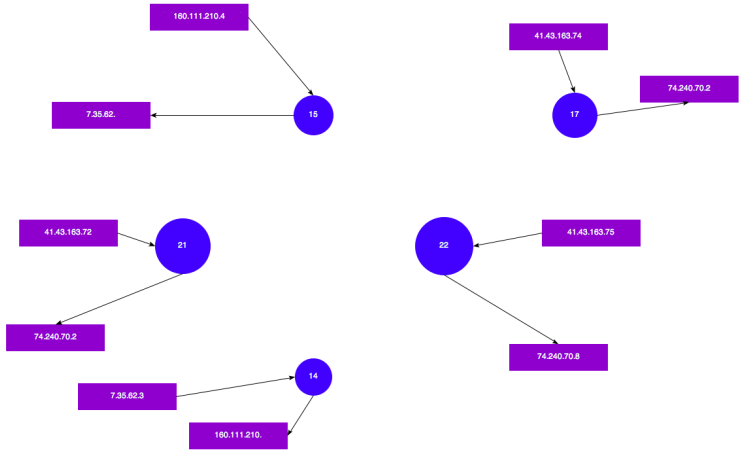

Figure 5: Visualisation of Streaming Data with Hyper Graphs

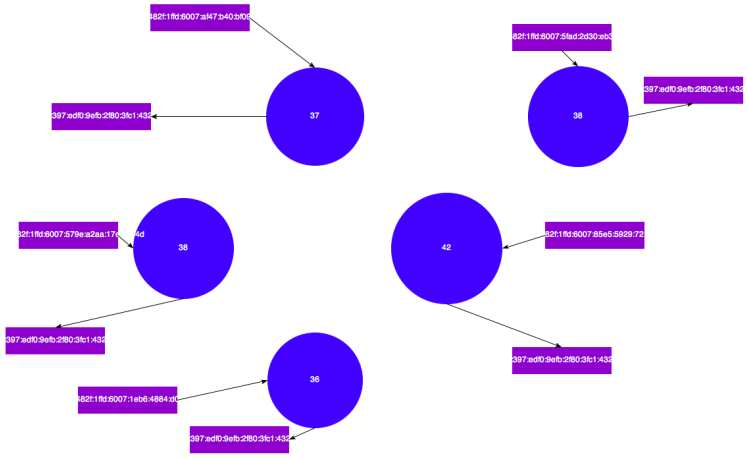

Figure 6: Visualisation of Streaming Data with Hyper Graphs

naive approaches. Three, one future work can be construct the complete system and invent new schemas.

References

[1] Glatz, E., Mavromatidis, S., Ager, B., and Dimitropoulos, X. Visualizing big network trac data using frequent pattern mining and hypergraphs. In *First IMC Workshop on Internet Visualization* (2012).

[2] Mozafari, B., Thakkar, H., and Zaniolo, C. Verifying and mining frequent patterns from large windows over data streams. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering* (Washington, DC, USA, 2008), ICDE '08, IEEE Computer Society, pp. 179–188.

[3] Zelle, J. M. *Graphics Module Reference*, 4.1 ed. http://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf, Fall 2010.