Network Flow Visualization
CS519: Scientific Visualization (Fall 2021)

Vincent Chung*
Department of
Computer Science
University of Illinois at
Urbana-Champaign

ABSTRACT

Computer networks facilitate the exchange of vast quantities of information between geographically disparate hosts. In order to optimize large networks for security, quality of service (application latency) and bandwidth utilization, the routing and packet forwarding behaviour of a network are configured and verified by studying flows through the network. In this project, a web-based network flow graph is constructed to visualize IP connection data collected at the Universidad Del Cauca across six days in April and May of 2017 [1].

Index Terms: Human-centered computing—Visualization—Visualization techniques—Treemaps; Human-centered computing—Visualization—Visualization design and evaluation methods

1 INTRODUCTION

In this project, a web-based network flow graph is constructed to visualize IP connection data collected at the Universidad Del Cauca across six days in April and May of 2017 [1]. An interactive graph of ongoing connections at user-specified points in time are rendered showing the rate and composition of data flows between hosts in the network.

The tool may be extended to be used for any collection of network data flows, presenting active network overlay topologies, and replaying historical events, for example such as day-night traffic and congestion cycles, and DDoS attacks.

The code implementation is organized under the `dataset/` and `visualization/` directories for the dataset preprocessing and web visualization portions respectively. These are described in the following sections.

2 DATASET

The dataset used for this visualization was collected at the Universidad Del Cauca across six days on 26, 27 and 28 April, and 9, 11 and 15 May of 2017 [1]. It consists of 3,577,296 entries created by the CICFlowMeter tool [2] monitoring a portion of the university's network. The flows are augmented by data from the ntopng tool [3], which performed Deep Packet Inspection (DPI) on sampled packets to infer the Layer 7 application protocol from the traffic data and flow patterns.

2.1 Dataset Preprocessing

To prepare for use by the visualization application, the raw dataset is filtered, sampled and compiled into JSON files to display network flows active at instantaneous timestamps (Fig. 2).

*e-mail: vwchung2@illinois.edu

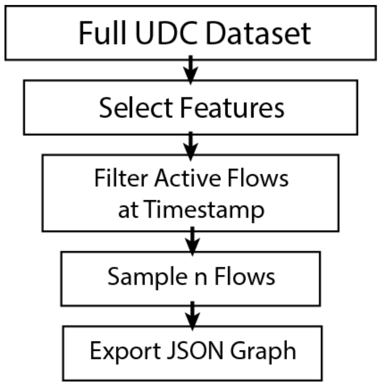

Figure 1: The network graph data preprocessing pipeline.

Preprocessing is implemented with the `preproc-data.py` script to generate a JSON network graph at a specific timestamp. The gen-all-graphs.py script uses the former to preprocess flow data at six timestamps of interest (morning and evenings) with four quantity of flows, and then generate a manifest file. The quantity of flows chosen is somewhat arbitrary, but are meant to allow the entire graph to fit into conventional web browser displays and fully render the network topology.

Table 1: List of fields extracted from the Universidad Del Cauca dataset. [1]

Attribute	Description
Flow.ID	Unique flow identifier.
Source.IP	Source IPv4 address.
Destination.IP	Destination IPv4 address.
Timestamp	Date and time of flow commencement.
Flow.Duration	Duration of the flow.
Flow.Bytes.s	The average data rate of the flow.
L7Protocol	DPI Layer 7 protocol code.
ProtocolName	DPI Layer 7 protocol name.

The Python Data Analytics Library (Pandas) [4] is used to ingress and filter the dataset. Out of the 87 features, 8 are extracted (Table 1). The flows are then filtered to select only the ones active at the instant of the timestamp (T_{inst}), using the following:

$$Flow.Timestamp + Flow.Duration \leq T_{inst} \tag{1}$$

Next, the flows are randomly sampled with up to $n = \{100, 300, 4000, 10000\}$ in each graph. There may be hundreds of thousands of flows active concurrently, which is too many to render in a single graph. Another alternative option would be to perform edge filtering to preserve those links with highest betweeness centrality, but the dataset provides no guarantee that flows form a connected graph, and the removal of the individual edge node flows means that those flows remaining will be aggregates.

Finally, a graph is assembled, the source and destinations become nodes, and their connection becomes edges. Nodes with 3 or more connections is arbitrarily classified as switches. The data rate and protocol information of the flow is embedded in the edge connection.

3 Network Flow Visualization

The network visualization is created with the ccNetViz [5] library, an interactive graph visualizer with JavaScript bindings. It was chosen for its ability to render using WebGL, which is necessary for performance with large graphs, and customizability.

The graph is hosted as a single-page application, using the Bootstrap [6] and jQuery [7] libraries for layout and interactive controls.

3.1 Graph Structure

The network graph has flow terminus as nodes (each labelled with an IPv4 address), and the flow connection as edges. This is not necessarily equivalent to a physical topology (layer 2), as the network can be virtualized. Nodes with 3 or more connections are arbitrarily classified as switches, since most hosts only record a single active flow. This method generally produces good results in this visualization. Switch and host (i.e. client) nodes are shown with different glyphs to distinguish them.

The graph is rendered using the `force` layout with gravity, charge and friction mechanisms, an algorithm similar or derived from the *Graph drawing by force-directed placement* paper [8].

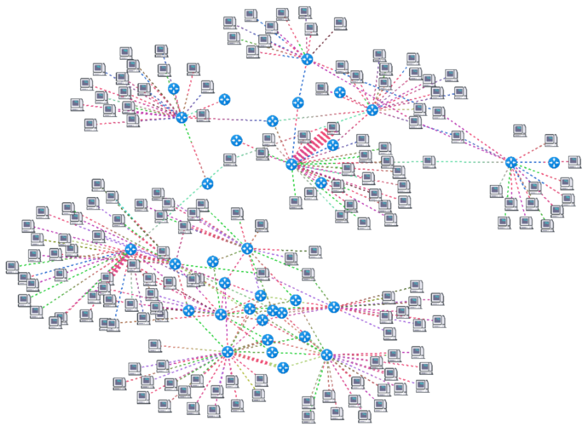

Figure 2: High-level graph view of network flows between clients and switches.

A sliding colour animation in the direction from the source node to the destination node indicates initiating direction of the flow. For example, a client loading a web page would be the source, and the server would be the destination. A server sending a push notification to a client would be the reverse direction. This is used in place of arrows to save screen space, as the large number of connections would inevitably occlude features in high-density regions.

3.2 Traffic Visualization

The magnitude of the traffic volume is represented by the width of the edges connecting nodes. The average data rate of the flow relative to the minimum to maximum of the entire sample set is linearly scaled to a value between 1 px and 6 px. Most flows are short-lived, but continuous sessions such as HTTP transfers and video calls will appear as large bands.

3.3 Traffic Decomposition

Each flow is augmented with an application-level protocol label from deep packet inspection, unknown and encrypted (with TLS tunneling) flows are opaque however. The protocol code is hashed to produce a pseudo-random RGB colour to label each flow connection edge, and a legend is rendered to allow the use to distinguish between the types of flows. Due to limitations of the graph library however, edge labels and tooltips cannot be rendered, where they would normally display additional context-sensitive details.

3.4 Interactivity

The graph visualization may be resized, panned and zoomed to show macro and micro level details of connections, and is performant to several thousand connections. However, this represents only a small fraction of the network activity in the dataset. An possible improvement is to aggregate network flows and its fractional compositions as the region of interest is expanded.

A control is present to select the number of network flows to sample and render, as the number increases, the disparate network subgraphs become connected. A series of labels with timestamps allows the user to select different instances in time to filter the network flows for the graph. This could also have been implemented as a slider with a continuous domain.

Due to technical limitations of the graph library, nodes and edges cannot be dynamically updated without redrawing the entire graph, so both the number of flows and timestamps were preprocessed offline to simplify the implementation. Ideally, a streaming algorithm can play back network flows continuously with low data overhead.

4 Conclusion

In this project, a web-based network visualization application was created, drawing network flows active at points in time as edges connecting client and switch nodes. Features such as the direction, magnitude and the composition of the flow are shown a visual properties, and interactive UI elements allow the number of flows and time attribute to be controlled.

The visualization my be improved by writing a custom graph renderer and data streaming backend that is not limited by programming model and extensibility of general purpose ones.

References

[1] J. S. Rojas, "Ip network traffic flows labeled with 75 apps," Kaggle, September 2018. [Online]. Available: https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps

[2] C. I. for Cybersecurity, "Cicflowmeter." [Online]. Available: https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter

[3] ntop, "ntopng." [Online]. Available: https://www.ntop.org/products/traffic-analysis/ntop/

[4] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[5] HelikarLab, "ccnetviz." [Online]. Available: https://github.com/HelikarLab/ccNetViz

[6] Bootstrap, "Bootstrap." [Online]. Available: https://getbootstrap.com/

[7] O. Foundation and jQuery contributors, "jquery." [Online]. Available: https://jquery.com/

[8] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380211102