5-2018

Dynamic 3D Network Data Visualization

Brok Stafford

Dynamic 3D Network Data Visualization

Brok Stafford
Department of Computer Science and
Computer Engineering
University of Arkansas
brokanxyz@gmail.com

ABSTRACT

Monitoring network traffic has always been an arduous and tedious task because of the complexity and sheer volume of network data that is being consistently generated. In addition, network growth and new technologies are rapidly increasing these levels of complexity and volume. An effective technique in understanding and managing a large dataset, such as network traffic, is data visualization. There are several tools that attempt to turn network traffic into visual stimuli. Many of these do so in 2D space and those that are 3D lack the ability to display network patterns effectively. Existing 3D network visualization tools lack user interaction, dynamic generation, and intuitiveness. This project proposes a user-friendly 3D network visualization application that creates both dynamic and interactive visuals. This application was built using the Bablyon.js graphics framework and uses anonymized data collected from a campus network.

1 INTRODUCTION

The majority of network visualization research has been done in 2D space and the most common approach is graph-based as shown in Figure 1. The 2D application OvlVis takes this approach and attempts to dynamically visualize Peer-to-Peer (P2P) networks [1]. The visuals generated by this application consist of nodes for network devices, connections between those devices, and traffic on those connections. Ever since the early days of networking, topologies have always been designed in this graph-based format, simply nodes and connections. However, the OvlVis tool creates these visualizations in real-time, showing new connections, disconnects, traffic and system alerts. The insight provided by an application such as OvlVis is invaluable to those who monitor networks. This straightforward and traditional approach has been proven to effectively allow both technical and non-technical individuals to better grasp networked systems.

Concerning 3D visualization tools, current research typically involves only static visualizations. A common approach to visualize network data in 3D is 2Dto2D visualization. A recent project done by Y. Okada uses this approach to visualize network intrusions [2]. He describes 2Dto2D visualizations as being composed of lines that are drawn from one 2D plane to another 2D plane in 3D space. This project uses the following four attributes for the visualizations: source IP, source port, destination IP, and destination port. Both the source and destination attributes are mapped to their own planes. When analyzing network data with this technique, patterns depicting port scans, DoS attacks, DDoS attacks and security hole attacks are easily identifiable. Of the few 3D network visualizations projects, another one uses the exact same approach as Okada's [3]. Although both of these projects take different approaches in gathering and preparing their data, each project's visualizations are almost identical.

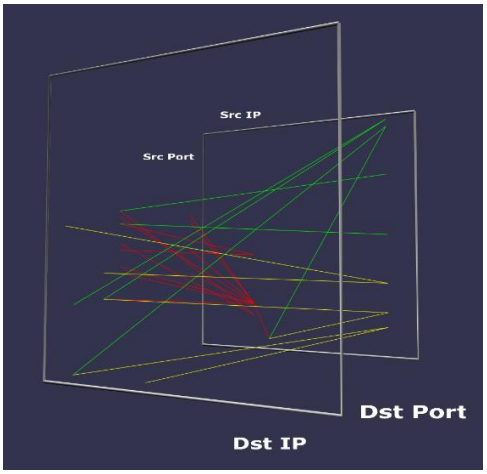

Figure 1: A typical 2D to 2D network data visualization.

Several years ago, the CyberSeer group from the University of Southern California performed research on a unique visualization tool [4]. Their approach was to take a large set of network data and convert it into time-based frequency data using spectral analysis. They then proposed to display this data in an immersive auto-stereoscopic 3D video and audio environment. This proposed environment displays video in 360 degrees around the operator. Therefore, spatial audio cues would notify the operator of events occurring outside of their vision.

There is a significant lack of 3D network visualization tools and research. Approaches used in 2D applications can be migrated to 3D space. Having an extra dimension to work with allows visualizations to become more detailed and complex. Due to the popularity of complex 3D video games, the average user should be comfortable navigating a detailed 3D visualization. As long as visualizations are not overly complicated, 3D space only increases the possibilities for displaying network data.

2 APPROACH

2.1 Overview

There is an obvious lack of research and tools that create 3D visualizations of network traffic. The 3D visualizations that do exist tend to be static, non-interactive, and not user friendly. The goal of this project is to create an application that fills that void. With this application users can dynamically generate visualizations based upon input parameters, they can interact with that visualization, and the application is intuitive. By incorporating visualization concepts that have shaped the success of 2D tools, this 3D application is more capable than other ones.

The visualizations created by the application are built using the Babylon.js framework. Bablyon.js is an open source project that allows for the creation of 3D graphics applications on the web by using the WebGL library [5]. Due to its compatibility with popular web browsers and no need for plugins or browser extensions, clients can easily use Babylon.js applications. The Babylon.js code is wrapped inside of a Node.js application. The Node.js application serves web pages to users and uses Socket.io to allow clients to talk to the server.

The bulk of the data processing is done on the Node.js server and then sent back to the client. The client will specify a single binary file to be sent to the server, then, based on the user's input, the server will use NFdump to generate all of the objects to be visualized [6]. The client will process the graphics itself. NFdump is an application used to analyze NetFlow and sFlow datagrams [6]. This project uses it to first preprocess incoming sFlow datagrams, then to anonymize those datagrams, and finally to generate visualization objects for clients.

2.2 Data Collection and Anonymization

The network data used for this project follows a lengthy preprocessing pipeline before it can be visualized and is shown in Figure 2. The traffic being collected originates from the University of Arkansas's on-campus network. One of the many distributions for the campus's network has been selected and the many switches that make it up are being continuously sampled. This traffic is sampled using the sFlow technology and has the initial form of sFlow datagrams. These datagrams are collected every five minutes and stored on a secure computer. The secure computer was initially configured so that incoming data will automatically be transformed, anonymized, and then sent off to a pickup computer. After this initial configuration, the machine was locked down, with access only available to the University of Arkansas's network team. The secure computer uses SFcapd, a NFdump tool, to capture sFlow datagrams. Upon capture, NFdump is then used to convert the files from sFlow to NFdump's own binary file format. Then the traffic is anonymized using another NFdump tool called NFanon. NFanon uses the CryptoPAn module to anonymize all IP addresses using the Rijndael Cipher. These anonymized files are then sent to the pickup computer using the Linux utility rsync. The visualization application is also running on the pickup computer and is ready to use the finalized data.

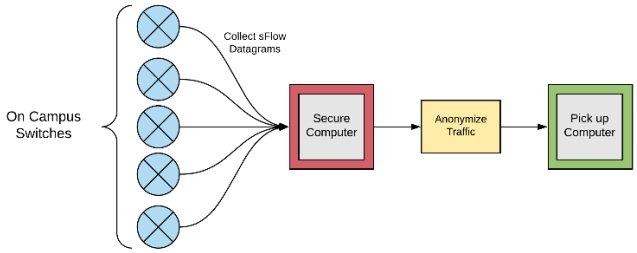

Figure 2: The network data collection and preprocessing pipeline.

The collected data is first in the form of sFlow datagrams. SFlow is a network traffic sampling technology that is standardized, scalable, and is low in computational cost [7]. sFlow datagrams contain data regarding protocols, layer 2, layer 3, and BGP [7]. Packets are gathered every 1000 number of packets. The sFlow datagrams are converted to NFdump binary files. These files contain the same data as they initially did; however, NFdump can now be used to aggregate and generate statistics on this data. For example, NFdump can display the top 100 destination IPs that have the most flows, or aggregate traffic based on source and destination ports. These statistics can be output in many different formats including CSV, biline, a user defined format, or many others.

2.3 Visualization Objects

Visualizations generated by this project are composed of four object types: IP nodes, tiers, port nodes, and connections. The first is an IP node as shown in Figure 3. These are spheres that represent a single IP address. Their size is based upon the amount of flows that are associated with that IP; the more flows, the larger the sphere. These nodes are all grey in color, except for the middle and most popular IP node, which is gold. IP nodes are placed on tiers. Tiers are flat outlines of circles drawn with white dashed lines. As new tiers are added to a visualization the greater their radius becomes. IP nodes are evenly distributed upon these tiers. Surrounding the IP nodes are port nodes. Port nodes represent the active ports for the given IP they surround. They are colored using a hash function that takes in its port number as input. Ideally, each port number will be given a uniquely identifiable color. The last object type is a connection. Connections are 3D lines that link port nodes together. Connections represent connections made between two separate IPs and allow users to see the flow of traffic. The Solar System model inspired the design of these objects and their placement. The golden IP node at the center is the sun, the other IP nodes are planets orbiting the sun, and port nodes are moons orbiting planets.

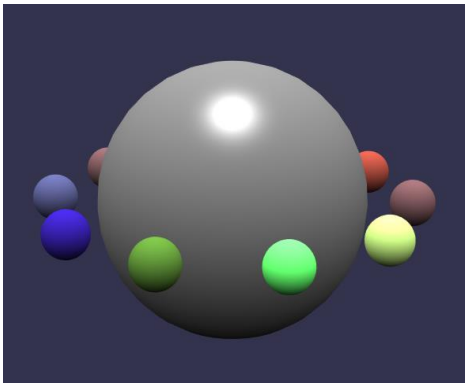

Figure 3: An example of an IP node with its surrounding port nodes.

2.4 Dynamic Visualization Generation

When this application is initially launched, there is not a visualization present. The visualization generation interface is the only element that is on screen as shown in Figure 4. This interface has two separate boxes for user input: essentials and dynamic options. The essentials box includes all of the necessary input required to generate a new visualization. Once the input has been configured, clicking the executeNFdump button will create the specified visualization. The input in the dynamic options box is used when a visualization is present. These inputs update the current visualization in real-time.

The essentials box is comprised of nine different inputs that determine the generated visualization. Starting from the top, the fileName field refers to the NFdump binary file that will be visualized. The nodesPerTier field defines the maximum amount of IP nodes that should be placed around a single tier object. The numNodes fields specifies the number of IPs from the NFdump binary that will be visualized. The minNodeSize and maxNodeSize fields define the range of size for IP nodes, which is based upon the amount of flows associated with a given IP. The ports and connections checkboxes determine if port nodes or connections will be drawn. The stat field refers to the statistic used by NFdump to generate the IP nodes. Instead of generating nodes based on IPs, users can specify a different field, such as source port, destination IP, or interface. There is a total of 35 statistics, however connections and ports will only be displayed when an IP statistic is chosen. The orderBy field specifies how the returned statistics should be sorted. The default is by flows, but there are 10 other options that NFdump has defined. Lastly, when executeNFdump is pressed, the client will send the Node.js server the inputs, the server will process the NFdump file and send the client a collection of object data, and finally the client will use that data to create the visualization.

The dynamic options box contains three inputs that will all update the currently displayed visualization. When any of these sliders are changed, functions on the client are called to clear the visualization and to then redraw it with the updated variable. The connVisible field determines how many connections are visible. The connHeight field will alter the height of all connections by a factor of its value. Lastly, the connRadius field defines the radius of the connection objects

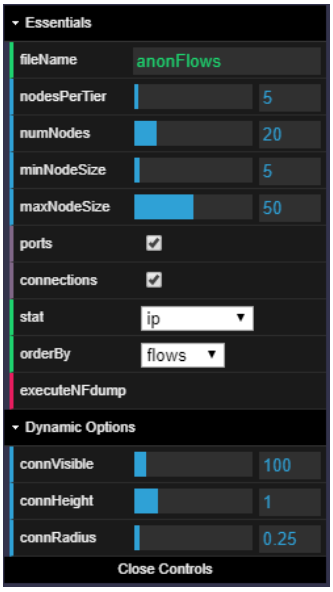

Figure 4: Visualization generation GUI.

2.5 Visualization Interaction

There are several techniques users can perform to interact with the visualizations. The camera can be adjusted with either the W, A, S, and D keys or by clicking with the left mouse button and dragging. Users can also left click on any object in a visualization, except tier objects, and view a window of in-depth data for that object as shown in Figure 5. When an IP node is clicked, its window will display its IP, flows, packets, and bytes. The flows, packets, and bytes include those incoming and outgoing. The percentage next to those three values indicates the ratio between the data for that node and the entire NFdump file of traffic being visualized. The windows for port nodes shows its port number, flows in, flows out, packets in, packets out, bytes in, and bytes out. Lastly, the windows for connections are comprised of source IP, destination IP, source port, destination port, flows, packets, and bytes.

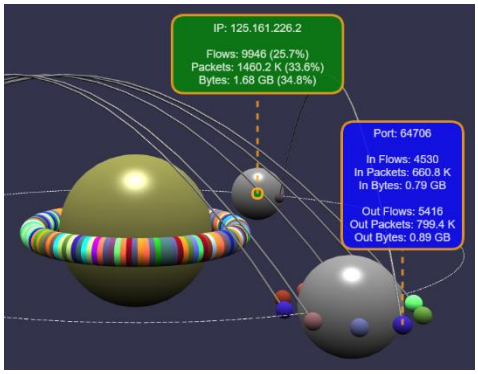

Figure 5: IP and port data windows.

3 RESULTS AND DISCUSSION

The goal of this project was to create a 3D network traffic visualization application that is dynamic, interactive, and intuitive. The final application incorporates all of these into its design, but with varying degrees of success. Since past 3D projects have done poorly in mapping out the field of 3D network visualizations, the design for this project is experimental. There is always room for improvement, however the design for visualizations in this application is a step in the correct direction.

These visualizations can be dynamically generated through the generation GUI. The input options within this interface are diverse and include a wide range of selectable values. Although there may be possible options lacking, the ones available allow for fine control of the generation. Users are able to interact with the visualizations through manipulating the camera's positon and angle, and by viewing in-depth windows of data for specific objects. Defining the level of intuitiveness of this application is difficult. The visualizations look clean and simple, even if the viewer does not understand what is being represented. A typical generated visualization is depicted in Figure 6. On the other hand, the same cannot be said for 2D to 2D visualizations. If the user has ever played a video game on the computer, operating the camera and navigating through this application will feel familiar. The overall readability and usability has been improved upon from the 2D to 2D visualizations. Therefore, this application is more intuitive to operate.

At its current state, this application can be used to successfully perform analysis on a given network. Analysts can plug their network traffic into this application and immediately find the hot spots in their network. They also have the ability to dig deeper into the data to find out why traffic is flowing in a particular way. However, users might have a difficult time if they are attempting to analyze multiple capture files.

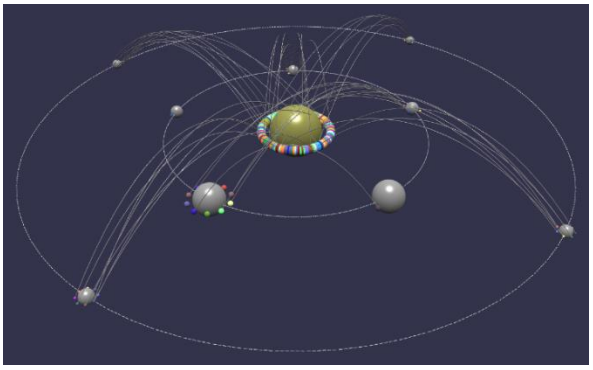

Figure 6: Overview of an entire visualization.

4 CONCLUSIONS AND FUTURE WORK

This project resulted in the creation of a sFlow collection and anonymization pipeline, and a 3D network visualization application. The data pipeline is completely automated and is continuously collecting traffic from the University of Arkansas network at five minute intervals. That data is anonymized with the NFdump library and sent over to a pickup machine where the visualization application processes it. Clients of that application can specify a visualization to be generated, the Node.js server will process the anonymized traffic, return it to the client, and the client will finally generate their desired visualization. Clients then have the ability to manipulate the visualization using dynamic options. They can also interact with the visualization by moving the camera and clicking on specific objects in order to receive detailed data on an object.

While the resulting application improved upon similar previous projects, there is still a need for improvements. The present application will only visualize traffic for a given static time frame. Having the ability to visualize traffic dynamically from a start time to an end time would help the analysis process. This type of visualization would have basic playback functionality, such as pause, rewind, fast-forward, etc. Adding animations to represent the flow of traffic and disconnecting and reconnecting IPs would increase the readability of a playback visualization. This type of playback visualization could support both historical and real time data. Interfacing this application with an Intrusion Detection System (IDS) would also increase the utility of this application. Analysts could pinpoint problematic timeframes on their network with an IDS and then visualize those periods.

ACKNOWLEDGMENTS

This work is supported by the Honors College and the IT Services Network Enterprise Systems group at the University of Arkansas.

REFERENCES

[1] K. Jünemann and J. Dinger, "OvlVis: visualization of peer-to-peer networks in simulation and testbed environments", 2008 11th communications and networking simulation symposium, New York, 2008, pp. 164-171.

[2] Y. Okada, "Network Data Visualization Using Parallel Coordinates Version of Time-tunnel with 2Dto2D Visualization for Intrusion Detection," 2013 27th International Conference on Advanced Information Networking and Applications Workshops, Barcelona, 2013, pp. 1088-1093.

[3] M. Coudriau, A. Lahmadi and J. François, "Topological analysis and visualisation of network monitoring data: Darknet case study," 2016 IEEE International Workshop on Information Forensics and Security (WIFS), Abu Dhabi, 2016, pp. 1-6.

[4] C. Papadopoulos, C. Kyriakakis, A. Sawchuk, and X. He, "CyberSeer: 3D audio-visual immersion for network security and management", 2004 ACM workshop on Visualization and data mining for computer security, New York, 2004, pp. 90-98.

[5] David Catuhe, David Rousset and other contributors (2013). Babylon.js v3.0, Apache 2.0 License. https://www.babylonjs.com

[6] Hagg, P. (2004). NFdump v1.6.17, BSD License. https://github.com/phaag/nfdump

[7] sFlow. 2003. Traffic Monitoring using sFlow. Retrieved from https://sflow.org/sFlowOverview.pdf.