**Network Visualization Project**

**Terry Wen**

**Lisa Marvel**

**Sean Morrison**

**1. Introduction and Background**

As part of the ever expanding age of information, data networking continuously grows in importance as the security of data becomes a bigger and bigger issue. With the quantities of data transferred across networks increasing day by day, the monitoring of such networks becomes increasingly difficult. The Network Visualization Project aims to tackle these issues of handling network traffic as well allowing such data to be presented in an easily usable and understandable fashion.

When utilizing available network tools such as WireShark, users lack direct control of the data they receive due to their only interaction being with the front-end output of data capture. Other visualizers tend to forfeit details of the data over pure aesthetic value of their illustrations, which creates issues of the programs ultimately being unusable for analysis. The Network Visualization Project aims to address these issues in its construction of a tool that optimally balances between content and clarity.

Because of the need for attention to detail, the Network Visualization Project is intended for use on the monitoring of small-scale networks to watch for specific issues or events in traffic. The primary goal of the project is to create an interface which can both help in locating potential issues or weaknesses in a network and illustrate an understandable model to a more untrained eye. In accomplishing this, the project will have provided a universal tool for the communication and clarity of network analysis.

**2. Design**

The Network Visualization Project consists of two parts: back-end data handling and front-end presentation. Data is first processed through Dshell, an open source packet decoding tool developed by ARL. This data, either streamed or input through a pcap file, is output in a readable JSON format as input into the front-end application of the project. This interaction of the user with the back-end interface allows for full manipulation of the data and ability to directly filter data to the user’s needs.

The data is then processed by the front-end application, which is coded using the Processing programming language. Here, the user will be able to visually interact with a timeline that encloses the duration of imported packet data. This playback presents the data visually through nodes and connections which can be manipulated fully by the user or automatically arranged by the application. The primary attributes of the data to be shown are:

* Source and destination IPs and ports
* Protocols
* Time
* Packet length
* Other details in packet header (TCP Flags, etc.)

The visualization is meant to illustrate macro activity through a network as well as the individual details of individual packets. The interface will also include other utilities such as playback tools and data filters.

Fig. 1 Design Model

**2.1 Dshell – Back-End Data Handling**

Dshell, or Decoder shell, is a shell script command-line framework used for network forensic analysis. Dshell processes existing pcap files and filters output information based on both pre-built and user-constructed modules. The Dshell portion of the Network Visualization Project involves the construction of an output module which will parse the necessary data from a pcap into a JSON format. The use of Dshell for this project allows the user to directly choose the modules to use for full control of the data portrayed in the visualization. Dshell provides the tools necessary to process and retrieve the desired information and is flexible enough to adapt to every individual’s needs, but will require more knowledge to operate to its potential.

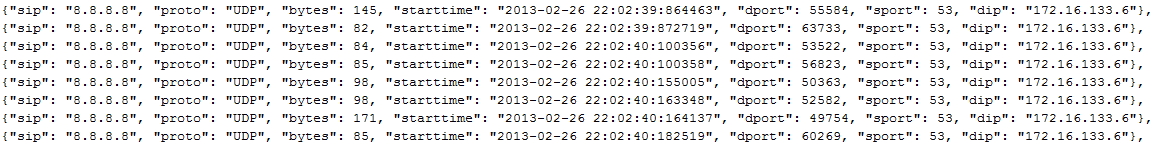


Fig. 2 Sample JSON Data

**2.2 Processing – Front-End Data Presentation**

The Processing application is a Graphic User Interface which first loads and processes the input JSON data into a visual. The JSON array, which contains up to thousands of packet objects with packet information as attributes, is loaded in using Processing’s Java-based JSON input methods. The packets are then added into individual arrays, generating the nodes and connections the travel between and through in the process to create whole arrays of data. The timeline is also generated based on the timestamps of the imported packet data, and the packet data is accessed through the individual states of the timeline.

The initial design of the visualization utilized a direct animated visual for the individual packets of an exchange, allowing for a full illustration of the amount of traffic on a connection between nodes. However, multiple issues arose from this, including the inaccurate portrayal of travel time for the data packets, which would travel faster than the eye could process clearly. This initial version utilized a timeline of visual states, where the visualization would be drawn to the timeline directly. The illustration ended up with inconsistencies, such as response packets being drawn while the initial packets were still portrayed as “travelling,” and packets being “frozen” while the timeline is paused when they were likely already at their destination (Figure 3). While the idea of viewing individual packet movements was appealing, the design was ultimately impractical and was readapted to a new model.

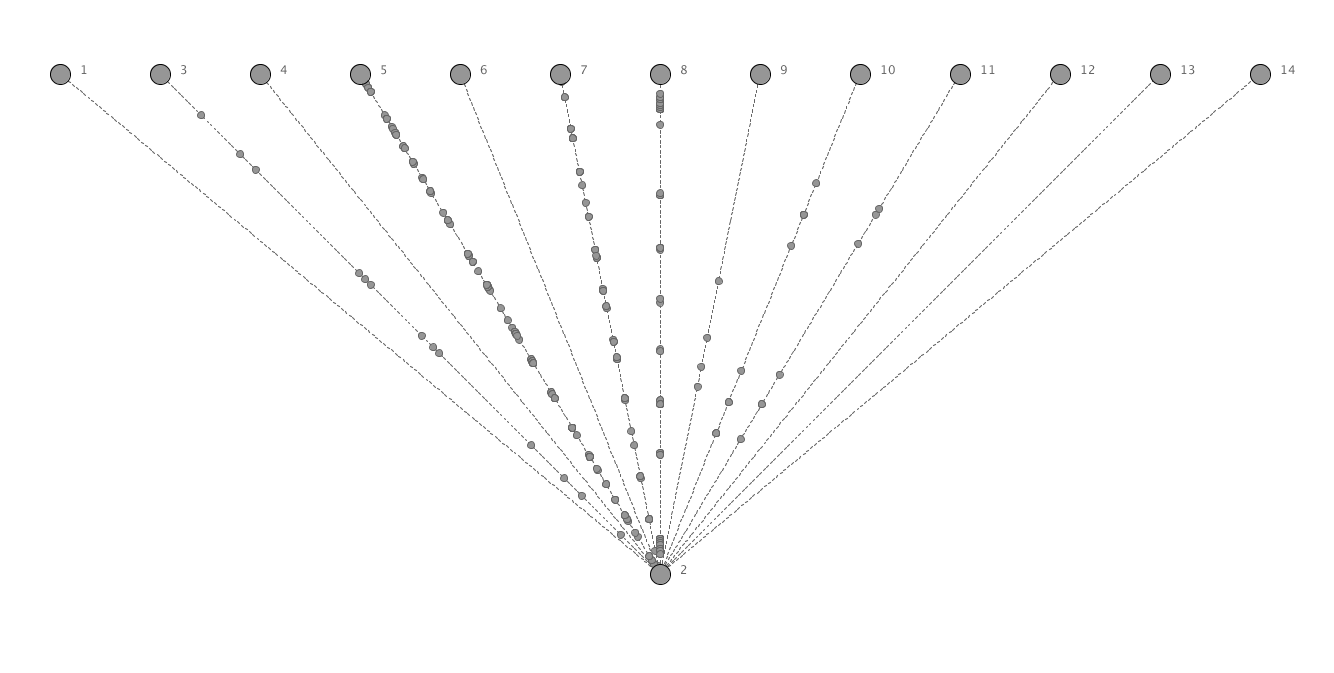


Fig. 3 Original Visualization Model

The new model utilizes flow visualization rather than individual packet visualization. Instead of animating individual packet movements, the visualization portrays general activity along a connection, which can then be selected to show more details about that flow of packets, including information on the individual packets. This model allows the visual to base itself on the data instead of the timeline directly, and the timeline instead influences the state of the data. By doing this, independent animations of nodes and connections can be manipulated far more easily as the model adapts itself to the data rather than the timestamp on the timeline. This allows for a simpler layout that also doesn’t sacrifice clarity or detail. In the screenshot below (Figure 4), active data transfer is denoted by a solid connection, and the currently selected connection is denoted by the highlighted connection.

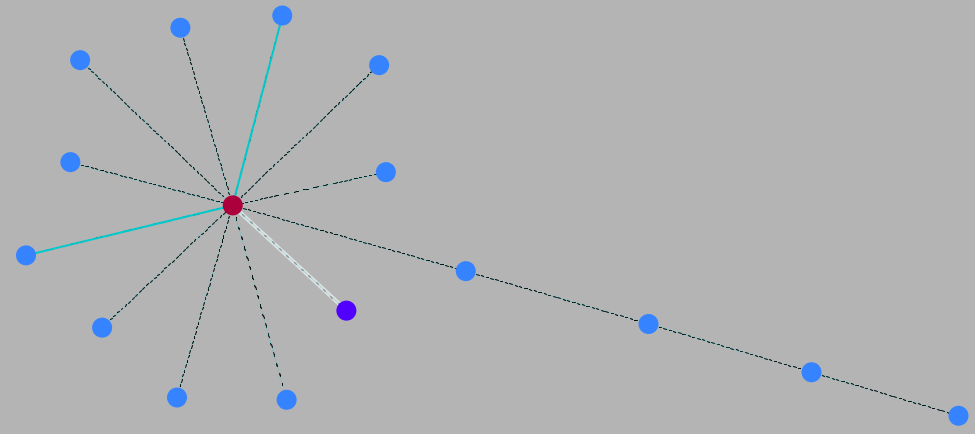


Fig. 4 New Flow-based Model

Using this new functionality, selecting individual packets for details also becomes much easier. As stated above, the selected connection is then illustrated in the interface as a smaller timeline as well as highlighted in the overall timeline, to allow viewing of individual packets and their sizes (Figure 5). The red line denotes the current point in time, and packets sent are either drawn above or below the timeline.



Fig. 5 Snip of connection timeline

This new model also allows for more flexibility in changing the layout of the visible graph, allowing individual nodes and connections to be visible only when they are actively sending or receiving data to reduce overall clutter. The GUI allows for easy manipulation of the individual positions of nodes so the user can customize the locations while the program also automatically balances the layout utilizing force algorithms.

Aside from the visual aspects of the GUI, selected nodes, connections, and packets also provide direct information including addresses and protocols (Figures 6). TCP and UDP is depicted in the visuals through different colors but can also be viewed in the attributes of selected objects. However, there is still much information to be incorporated into these text panels, for at the moment only the most important are included in the application.

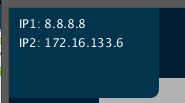
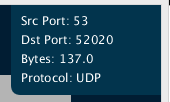
 

Fig. 6 Connection and packet information

**2.2.1 Visualization Operation**

At the moment, opening the application is a primitive process as the application has not been exported and runs through the Processing IDE. After retrieving the JSON file through Dshell, it should be named data.json and placed into the data directory in the application folder. Then the application file NetVis.pde should be run through the Processing IDE (Figure 7).

From here, there are multiple controls to interact with as well as the visualization itself. The timeline can be manipulated freely similar to a video player, and there are two timers, one for direct timestamps and one for the duration of the playback only. On the right side, there is a slider that manipulates playback speed from 0.1x up to 100x speed. There are also multiple buttons which manipulate the layout, with two primary functions:

Lock – Keeps node visible at all times, even when inactive

Anchor – Keeps node anchored at its current position, unaffected by the automatic position adjustment.

There is one final button labelled ‘Start of Flow’, which simply moves the timeline cursor to the beginning of the currently selected connection. Nodes and connections can be freely selected individually, and packets can be selected while a connection is active.

Along with the button functionalities there are also a variety of keyboard shortcuts:

Space – Play/Pause

L – Lock current node/all visible if no node is selected (Shift+L for all)

A – Anchor current node/all visible if no node is selected (Shift+A for all)

F – Move to start of selected flow

Future planned functionalities not yet implemented are covered in the “Future Work” section of this report.

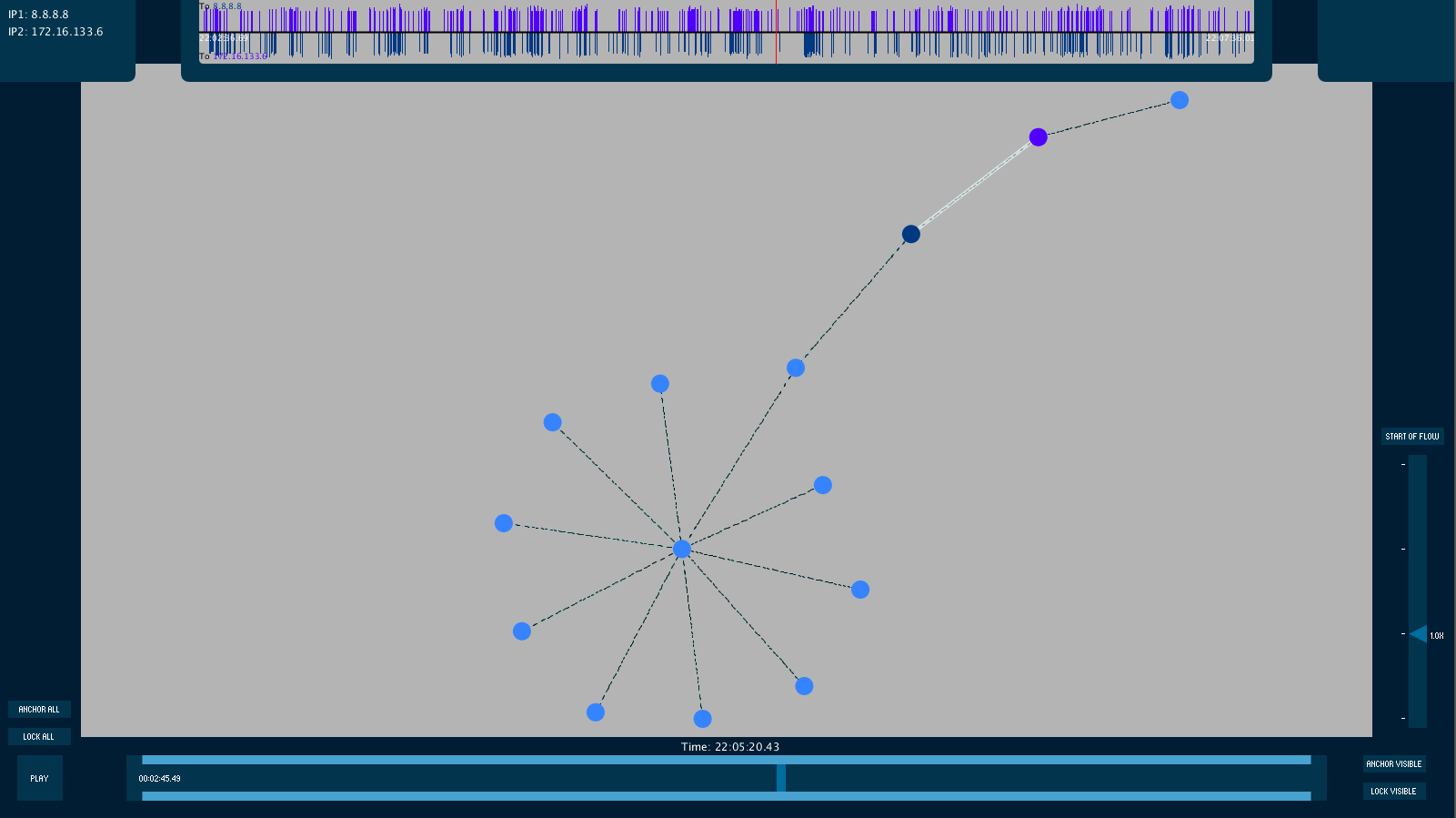


Fig. 7 Entire Application

**3. Conclusion and Future Work**

Millions utilize data networks every day in the modern era, but few are truly aware of what really happens in these systems. As network size and usage continue to grow, the chances of error and threats become more and more pertinent. Nowadays, even the common household should be able to monitor and make full use of their home networks to avoid exploitation and other possible issues. The Network Visualization Project seeks not only to provide a helpful tool for experienced analysts but also to promote awareness of the importance of how networks are structured and operate.

Though the base of the Network Visualization Project is complete, there are still much to expand upon. As stated in the original goals for the project, aesthetics should not sacrifice detail in an interactive interface. There are still many details in the packets which are still not presented in the final application, which currently functions as a large-scale visualizer. A primary attribute of packets to highlight are TCP flags to help capture more detail into what is occurring in each of the flows portrayed in the visualization. Other issues in the layout lie in the simplicity of the graphs: while the clarity is important, a difficult issue to address is how to distinguish similar nodes without having to look into the details.

These issues can hopefully be accounted for with future improvements to the user customization and interaction, which could utilize color-coding and node filtering as well as other ways to present information such as node sizing and clustering. Other ideas discussed included data exportation which could allow for shared visualizations that include highlighting, annotations and user-created layouts to best communicate analysis between users.

The application will likely eventually be released as a standalone application that doesn’t require the IDE to run, or be incorporated as a web application using Processing.js, a Javascript framework for Processing. All of these functions can be built off of the current base application, which is anticipated to be released as open-source to enable the innovation of these further possibilities.

**4. References**

Dshell. [accessed 8 June 2015]. https://github.com/USArmyResearchLab/Dshell

Processing. Documentation; [accessed 3 June 2015]. https://processing.org /reference/

Marchi, Lorenzo. “Force Directed Placement” [accessed 9 July 2015]. http://openprocessing.org/visuals/?visualID=177

ControlP5. Documentation; [accessed 20 June 2015]. http://www.sojamo.de /libraries/controlP5/reference/index.html

WireShark. [accessed 3 June 2015]. http://www.wireshark.org