Visualization of Network Traffic Using Wireshark Data
Semester Project

Abstract

Network security is a subfield of cybersecurity that is concerned with the detection, analysis, and protection of computer networks. Many tools exist that provide analysis with information about the traffic on arbitrary networks. The increase in the amount of data being generated by users necessitates that analysts use a wide variety of approaches for comprehending that data, including the use of visualization. In this project we will develop visualization support system for processing data from a widely used network traffic analysis system.

Introduction

A packet analyzer is an important tool for monitoring traffic being sent over a network. Example tools include Wireshark [1] and tcpdump [2]. Often referred to as "network sniffers", these tools capture information about messages sent between various nodes in a network.

A network can be considered a *graph* in that there are nodes/vertices that are interconnected by edges/links. The edges between nodes can be either physical or virtual in the sense that there is physical cabling that can be used to connect nodes versus the use of logical connections that between nodes that are achieved through the combination of protocols in the TCP/IP protocol stacks. In addition, with the advent of wireless networks, no longer do network nodes need to be physically connected, although, in a sense, wireless routers and access points do act as the physical gateway into a network.

Wireshark, and its command-line analog tshark, captures data in as close to a real-time fashion as possible and presents that information to a user in a stream of output. Indeed, the data that is captured can be difficult for a human reader to comprehend due to the volume and nature of the data being displayed. If any of that data is anomalous, as can happen when a network is undergoing cybersecurity attacks such as distributed denial of service (DDOS) attacks, trying to use Wireshark output as a diagnostic tool can be prohibitive.

At Tennessee Technological University we are conducting some research on the creation of visualization tools that display anomalies in graphs, including network graphs. In this project, you will be developing software that can be used to read, process, and visualize network data as part of a larger project called *Vizshark.*

Figure 1 shows an example of a graph that was generated in a live run of a preliminary solution of this project. As shown in the graph, each node is labeled with an IP address that corresponds to a node in a network (in this case, a home network). Two nodes in a network are connected if a message between two node is detected by tshark. Consider, for instance, the following sequence of data generated by tshark in comma separated values (CSV) format:

```
"5","1.436394000","UDP","17","192.168.68.105","192.168.68.255",,,"889","889"
"6","1.536100000","ARP",,,,,,,
"7","1.774322000","TLSv1.2","6","192.168.68.105","52.114.159.112","55358","443",,
```

The data consists of the following:

- Sequence Number
- Relative time from start of capture
- Text representation of the protocol used in the transmission
- Protocol code

- IP address of the source of the message
- IP address of the destination of the message
- Port number of the source (TCP)
- Port number of the destination (TCP)
- Port number of the source (UDP)
- Port number of the destination (UDP)

Based on the information provided in the data, only sequence numbers 5 and 7 would be rendered since information concerning the message in sequence number 6 is missing. The messages in 5 and 7 indicate that UDP is being used to transmit a message (5) from 192.168.68.105 to 192.168.68.255 on ports 889, and TLSv1.2 is being used to transmit a message (7) from 192.168.68.105 to 52.114.159.112 on ports 55358 and 443, respectively.

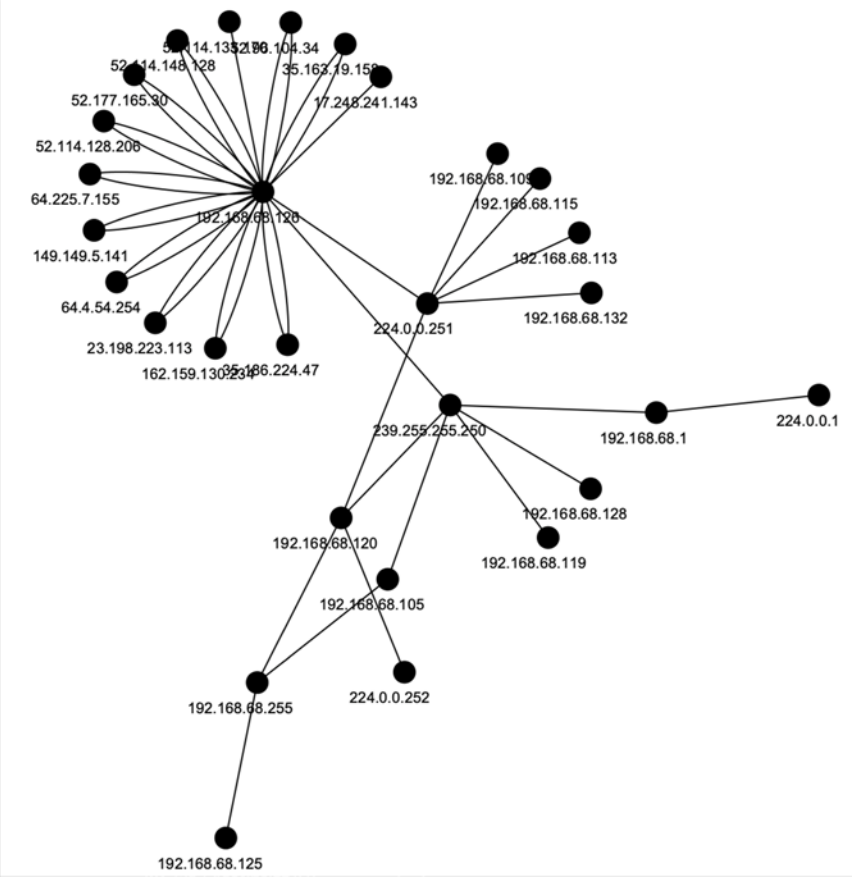

Figure 1 Vizshark Graph

The sequence found above results in the graph as shown in Figure *2*. As more data is read from the network, the graph will be dynamically generated resulting in a graph similar to that shown in Figure 1. The graph shows, primarily that many messages are one-way. However, the appearance of two graph edges between some nodes indicates that some nodes engage in two-way communication. At present, the graph does not show communication over unique ports, but the availability of port information does provide an opportunity

to display many more edges in a graph according to the multiplicity of messages that may be occurring between any two arbitrary nodes.

Key Concerns

The visualization system has three major components that govern its use: *data source generators, graph management,* and *visualization.* The *data source generators* are used to interface the system to sources of network messages. In general, these generators are either based on *playback* or *live-streaming*. The *data management* component provides an abstraction for graphs using an external library called *GraphStream.* Finally, the *visualization* component (also based on GraphStream) provides support for rendering graphs including creation of programmer-defined layout algorithms and sprite animation.

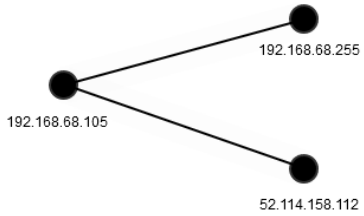

Figure 2 Graph of Sequence Number 5 - 7

Data Sources

- Playback sources: One way to use Wireshark is to run the system and then save a capture file (alternatively, a user can run tshark and redirect the output to a file). Regardless, the system we are building needs support for loading playback sources.
- Streaming sources: We envision the ideal use case for this system to be a streaming visualization that directly reads the output from tshark to capture data. A consequence of streaming a live source is that it may be necessary to support starting, pausing, stopping, and saving data read from a live streamed source.
- Data extraction: Regardless of data source, data needs to be extracted, parsed, and otherwise processed.

Data Management

- Graph Library: The system we are developing will use an API called GraphStream. This API contains support for managing graph-based structures, applying graph algorithms, and visualizing data: https://graphstream-project.org/

Visualization

- Support for animation: The GraphStream library provides support for animating message passing in a network. The animations provide a means for visualizing frequency of messages being passed between nodes in the network.
- Layout of nodes: For random or arbitrary graphs, dynamic layout algorithms provide a general approach for displaying a graph in a display. Network graphs, in general, have well-known topologies (i.e., layouts). We are interested in creating layout algorithms that mix the static

knowledge with dynamic algorithms to provide a means for better managing a network visualization.

Project Phases

The project will be conducted in two major phases: *concept-initiate* and *iteration-construction*. In the *concept-initiate* phase you will identify project requirements, create models, and prep your development environment to support your implementation activities. In the *iteration-construction* phase of the project you will iteratively develop software for the project by adding support according to the requirements developed during the *concept-initiate* phase. The schedule of the phases of the project and the individual iterations contained therein are defined below:

Date (Midnight)	Phase	Description
January 29	Concept Initiate 0	User stories
February 12	Concept Initiate 1	Use case diagrams
February 26	Concept Initiate 2	Class diagrams for Vizshark
March 12	Iteration 0	Project environment setup and initial execution demonstration
April 2	Iteration 1	Data source implementation: CSV support
April 16	Iteration 2	Streaming support and animations
April 30	Iteration 3	Graph layout algorithms

Concept Initiate 0

User Stories

In the initial phase of the project, you will create user stories based on the description provided while also generating ideas of other potential features for such a system. The user stories that you create should be specified in the following form:

As a <user type>, I want <desired feature>, so that <outcome>

You must also include, with each user story, a description of how you would expect to be able to observe this behavior in the system. Your source of information for the user stories you will identify are given in the description provided above as well as any discussion sessions we conduct in class. A template for your user stories is provided below.

User Story Template

Use the following user story template for *concept-initiate 0*. Replicate this as many times as is necessary.

User Story Name: <Short phrase for the user story>	
As a:	<user type>
I want:	<feature>
So that:	<outcome or reason>
Description	<description of how the user story / feature would be observed and tested in the system>

Submission

Submit your user stories as a PDF document only. Microsoft Word, LibreOffice, or any other native word processing format will not be accepted.

Rubric

Completeness: You will be graded on the completeness of your turn-in. In particular, you must analyze the description and attempt to identify as many features as you can that are relevant for the project. Note, there is no upper bound on the number of user stories you can define, but there is a lower bound that *will not* be specified.

Correctness: You will be graded based on your adherence to the format of the user story template and on the accuracy of the user stories with respect to relevance to the described project, including your description of how you might expect to observe the user story / feature in the system.

Points

The assignment is worth 20 project points.

Concept Initiate 1

Use Case Diagrams

In our initial step on this project, we did some work to try to identify user stories for the *Vizshark* system. The baseline set of user stories has been uploaded to the course iLearn site and should be used as a starting point for completing the next activity in the project.

For Concept Initiate 1, you will be creating use case diagrams for the *Vizshark* system. The baseline user stories fall into a few categories:

- Data Sources: stories that identify where the data should be drawn from (either playback or live-streamed)
- Data Management: stories that identify what information is to be captured and how it should be represented in the visualizations
- Visualizations: stories that identify how the information should be presented to the user (i.e., via animation and according to user-defined layouts)

As you did in Laboratory 2, you will use StarUML to create your use case diagram(s) for Vizshark. Note that the following are true about the overall system:

- Tshark is an external actor for the system and should be represented accordingly
- The Data Sources can be executed and generated by users that are different than the user of Vizshark visualization component

Submission

For your turn-in, you should use either *Snip and Sketch* for Windows, ⌘-Shift-5 on Mac, or Shift-PrtSc in Linux to do a screen capture of your model from StarUML. Copy and paste the image to word or some other word processing suite and generate a PDF. You must also include the user stories that you included in your model as an appendix using the same format from Concept Initiate 0.

Rubric

Completeness: You will be graded on the completeness of your turn-in. In particular, your submission must at the very least include the user stories provided in the baseline set. However, it is also expected that you will include other stories in your diagram.

Correctness: You will be graded on the correctness of your use case diagrams. In particular, your submission must correctly represent use cases, actors (both interactive and external actors), and partitioning of the stories into appropriate subsystems.

As a reminder, the project schedule is as follows:

Date (Midnight)	**Phase**	**Description**
~~**January 29**~~	~~Concept Initiate 0~~	~~User stories~~
February 12	***Concept Initiate 1***	***Use case diagrams***
February 26	Concept Initiate 2	Class diagrams for Vizshark
March 12	Iteration 0	Project environment setup and initial execution demonstration
April 2	Iteration 1	Data source implementation: CSV support
April 16	Iteration 2	Streaming support and animations
April 30	Iteration 3	Graph layout algorithms