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.

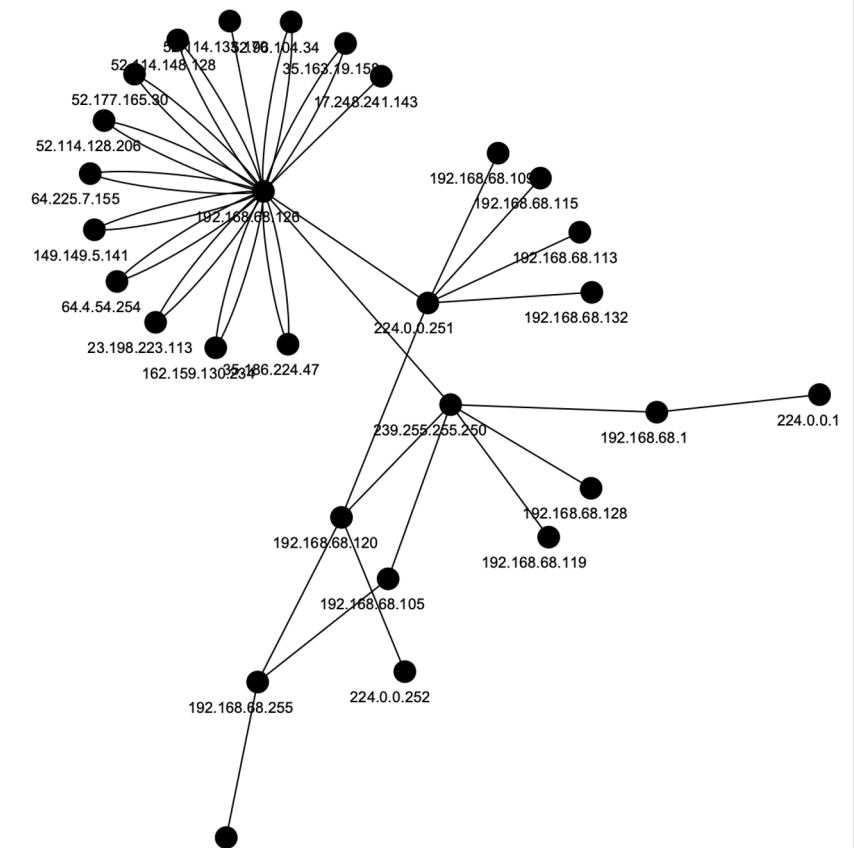


Figure 1 Vizshark Graph

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.

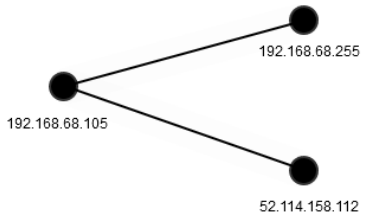


Figure 2 Graph of Sequence Number 5 - 7

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

Concept Initiate 2

Project Background

In earlier iterations of the project we developed user stories and created a use case diagram for the VizShark system. In the reference solution, shown in Figure 3, the Use Case Diagram is partitioned into packages called **GeneratorServer** and **VizSharkClient**. The partitions demonstrate that the operations are localized according to an implementation that focuses on data acquisition and data visualization. The reference solution shows the "**<<includes>>**" relationship although it is acceptable to have them appear in the model as separate use cases. The use cases are based specifically on the user stories provided in the posted solution from Concept Initiate 0.

The use case diagram identifies the following features for the system:

- Read data: The system should have the ability to use TShark to read data from a network, including
  - Live Data
  - Captured Data
  - Select specific fields to display

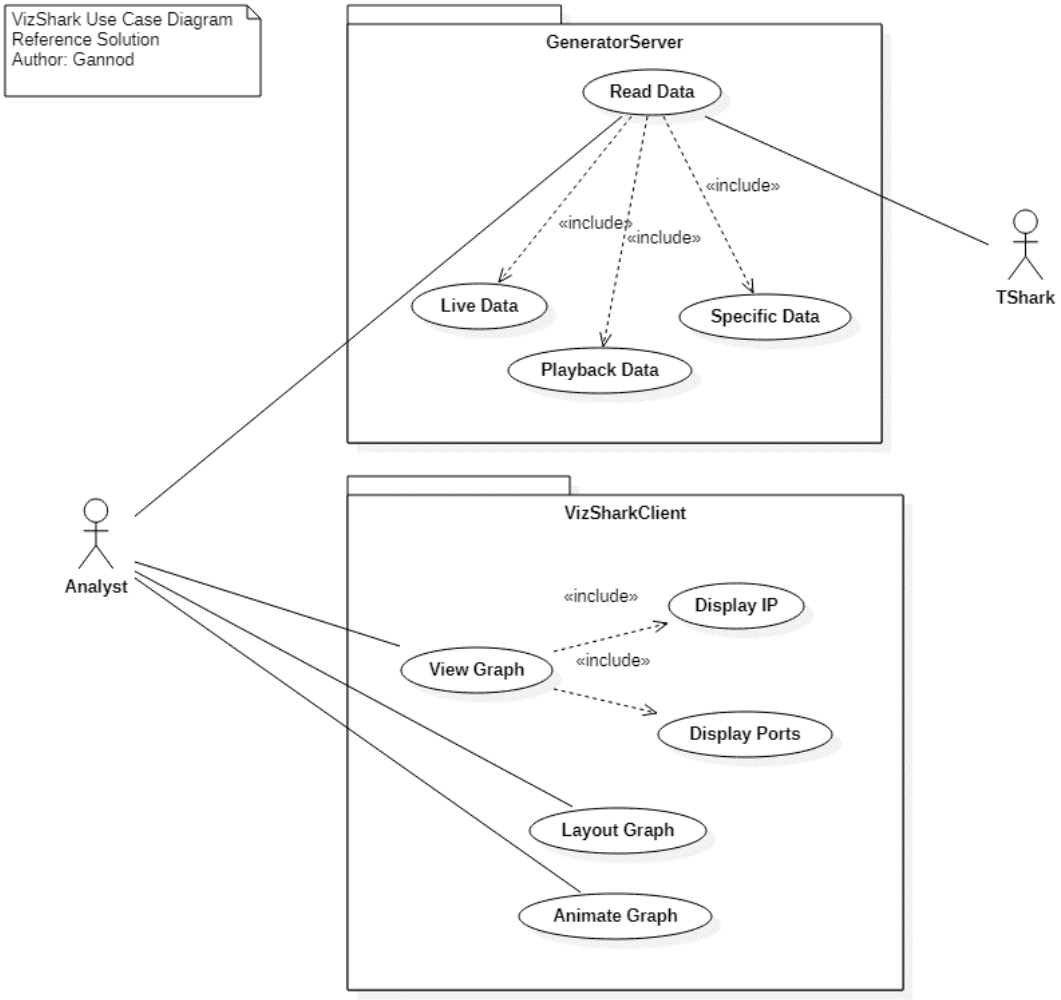


Figure 3 Use Case Diagram Reference Solution

- View network data as a graph: The system should display network traffic as a graph that shows messages being passed between nodes in the network.
- Layout a graph: The graph should be displayed using different layout algorithms.
- Animate the graph: Visualizations should be animated to provide emphasis of messages being sent in the network.

Architecture

The architecture for VizShark is shown in Figure 4. The system is partitioned into two parts: the **VizSharkClient** and **GeneratorServer**. They communicate using a Java networking solution known as *Java Remote Method Invocation* or Java RMI. In this architecture, a single client (i.e., VizSharkClient) can connect to multiple servers (as shown by the relationship between **VizSharkClient** and **GeneratorServer**).

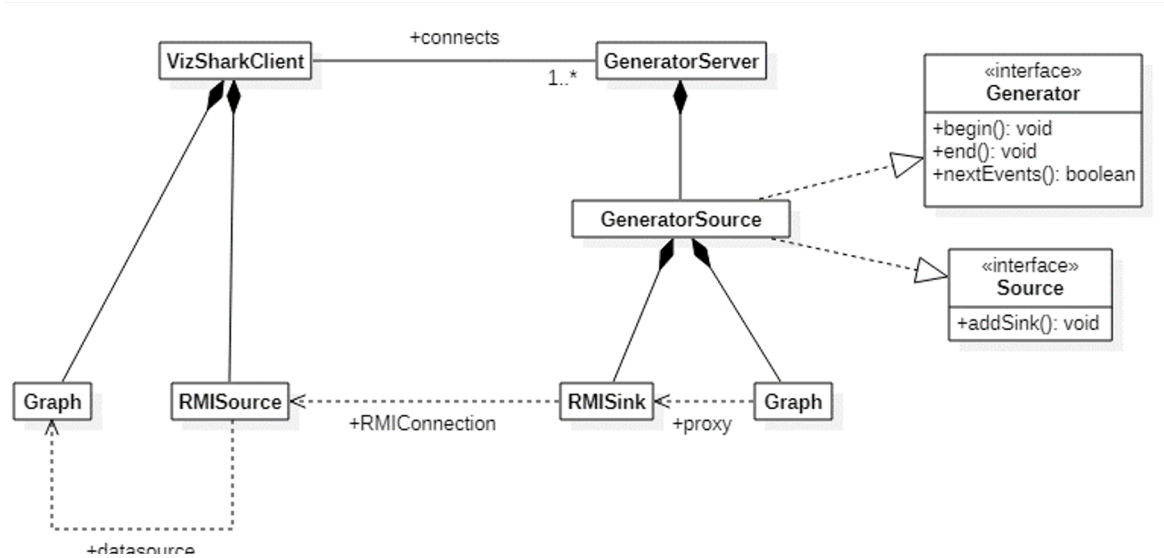


Figure 4 VizShark Architecture

GeneratorServer

The GeneratorServer is the application that manages reading data and sending information to be visualized to the VizSharkClient. The GeneratorServer uses a GeneratorSource to consume data that is then communicated to the client using the RMISink through an RMIConnection.

VizSharkClient

The VizSharkClient is responsible for all visualization tasks, which is performed by a Graph object. The RMISource object is used to associate the Graph object with expected data source for the system. Through Java RMI, the RMISource receives messages from the GeneratorServer (and specifically RMISink methods). The

Project Assignment

For Concept Initiate 2, you will be modifying the design shown in Figure 4 to incorporate the features identified in the initial phases of this project. Specifically, we need to add classes to the model that capture the following ideas:

- **GeneratorSource**: The GeneratorSource class is intended to be a placeholder for different versions of data readers that can either handle live data or playback data (from CSV files, for instance). As such, you should extend the model using the inheritance or subclass relationship to capture this idea in the model. Notice that the GeneratorSource class implements the *Source* interface and *Generator* interfaces, which are defined in the GraphStream library. The definition of those interfaces can be found at the following Javadoc links:
    - Source (gs-core: https://graphstream-project.org/gs-core/)
    - Generator (gs-algo: https://graphstream-project.org/gs-algo/)

  The model in Figure 4 shows the relevant methods for those interfaces.

  You will add two classes to the model that are subclasses of the GeneratorSource class:

    - *CSVGenerator*
    - *TSharkGenerator*

  Each should be defined as subclasses of GeneratorSource. You will need to spend some time thinking about how to incorporate RMISink and Graph appropriately in the model as to minimize the relationships drawn in the model.

- **RMISource** and **Graph**: The RMISource and Graph classes use a *pipe-and-filter* architecture that allow you to insert any number of *pipes* in-between them in order to transform data received from the GeneratorServer. The animation and layout features will be implemented using this approach by creating two classes (*SpritePipe* and *LayoutPipe*) implement the *Pipe* interface. **You must add two classes to the model that show the new pipes in the relationship between the RMISource and Graph classes.**

Note that the classes you are adding (CSVGenerator, TSharkGenerator, SpritePipe, LayoutPipe) represent the classes that you will eventually be implementing during the *Construction-Iteration* phase of the project. Upon completion of this assignment, you will have a class diagram that specifies the structure of the system.

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 document or some other word processing suite and generate a PDF.

Rubric

Completeness: You will be graded on the completeness of your turn-in and the incorporation of the four classes in the model.

Correctness: You will be graded on the correctness of your class diagrams including the proper use of the relationships of association, inheritance, implements, and other similar relationships.

Submission format: You will be docked points for not adhering to the submission standards described above. Specifically, you must turn in a PDF document of your model as a single page.

This assignment is worth 30 points (completeness – 10, correctness – 15, submission – 5).

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

Iteration 1

Project Background

The first half of the project focused on creating an understanding of the application to be built. The design we created used several object-oriented concepts including composition (creating objects using other objects), inheritance (design of classes through "inheriting" attributes and methods from a more general class), interfaces (specifying the contract for what methods a class should implement). Specifically, the design (as shown below) provided the structure for a client application (VizSharkClient) and a server application (GeneratorServer).

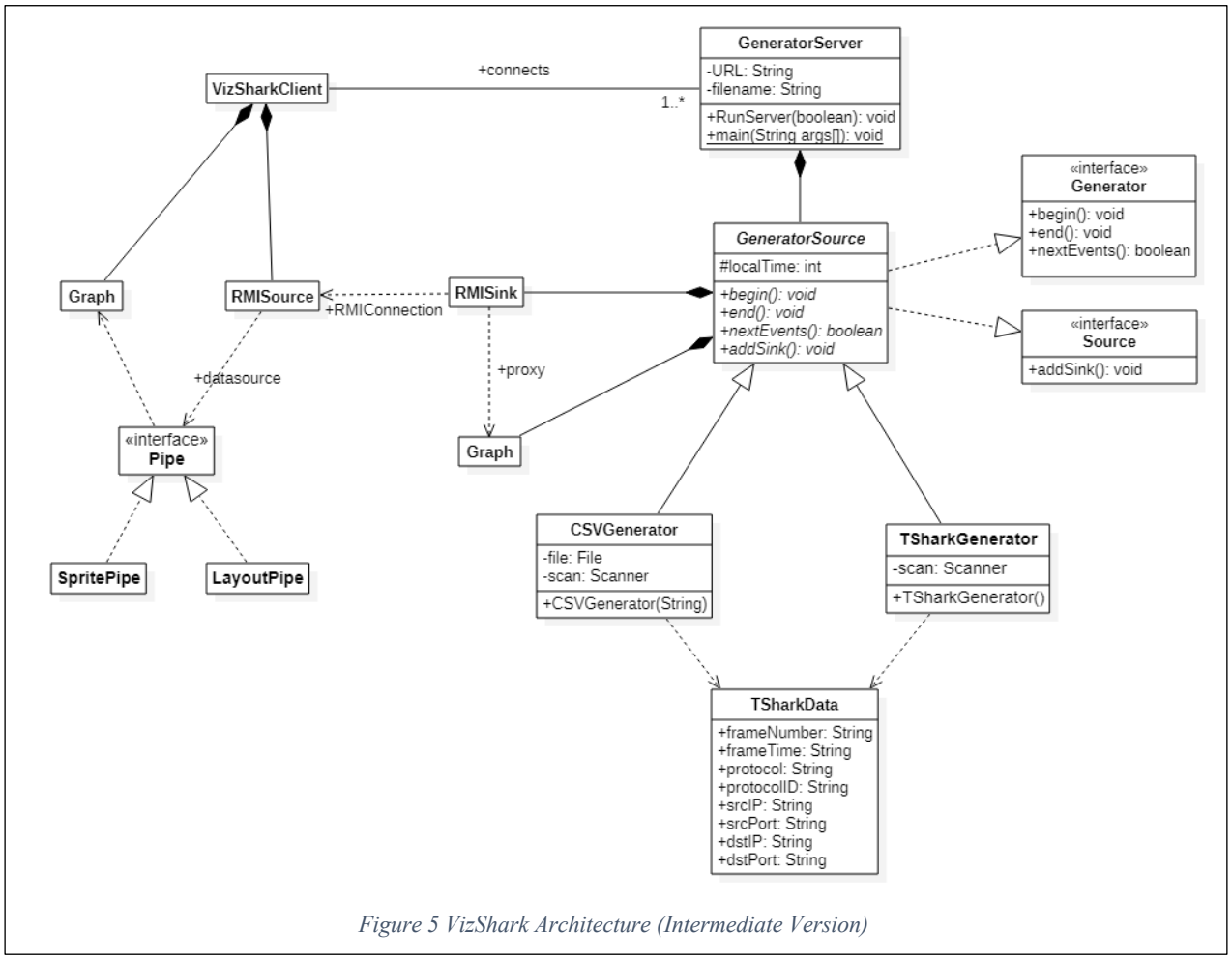


Figure 5 VizShark Architecture (Intermediate Version)

For the next stage in the project, we will be developing a simple version of the system as a first step towards creation of the full application. In particular, you will be implementing the application for the design shown in Figure 6. In particular, you will be creating an application called "VizSharkJunior" that is composed of a Graph and a comma-separated values (CSV) parser that reads TSharkData objects from a file.

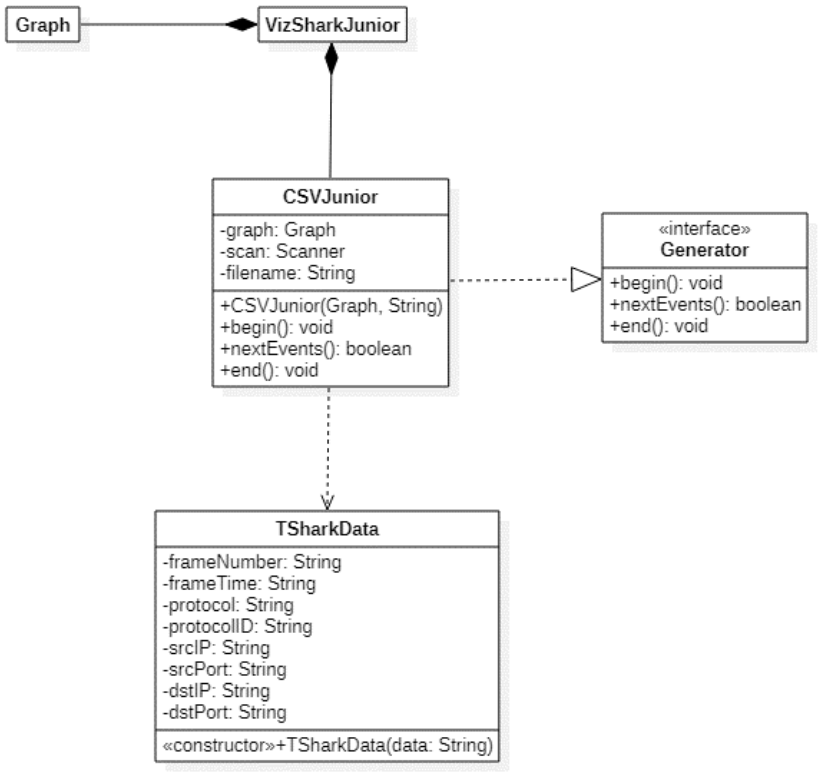


Figure 6 VizSharkJunior Architecture

CSVJunior

You will implement the CSVJunior class, which is responsible for opening and reading a CSV file that contains TShark formatted data and writing that data to a Graph object. The CSVJunior class has been provided to you save for the implementation of four methods:

- `public CSVJunior(Graph graph, String filename)` – constructor that takes a Graph object and String as input. Node and Edge objects should be added to this graph. The String is the filename for the input data.
- `public void begin()` – instantiates the CSV processing by opening the file and instantiating the Scanner object to be used by the nextEvents method
- `public boolean nextEvents()` – processes the data by using the scanner object (e.g., `while (scan.hasNext())`). This method returns true if at least one entity (a Node or an Edge) is read and added to a graph.
- `public void end()` – ends processing by closing the Scanner object by calling the `close()` method.

Laboratory 04 presented the meta-data / codex for data generated by the TShark application and included having you develop a class called TSharkData. The basic idea for handling this data is this:

- Read a line of data from a file using a Scanner object
- Pass the line of data to the constructor for TSharkData to parse the network data and use the resulting object to gain access to the *srcIP* and *dstIP* for the network communication

- Draw the data in a graph by adding *srcIP* and *dstIP* as Node objects, and adding (*srcIP, dstIP*) as an Edge object. (The in-class exercise "GraphDemo" provides the methodology for creating a Graph and adding Node and Edge objects). When creating the id for the nodes and edges, simply use the *srcIP* and *dstIP* strings for the node *id* and "*srcIP*:*dstIP*" as the edge id.

When adding data to the graph, make note of the following:

- The `getEdge(String id)` and `getNode(String id)` methods in the Graph class will return an edge or node, respectively, that has the given id.
- Attempts to add nodes and edges to the graph will fail if said node or edge already exists, as such, you should use the above methods to first search for either before adding them to the graph.
- When adding an edge to the graph, set the initial "label" for the edge to the integer value 1. If an edge already exists, read the "label" attribute using

  ```
  Integer weight = (Integer) e.getAttribute("label")
  ```

  and use

  ```
  e.setAttribute("label", weight + 1)
  ```

  to increment the weight of the edge.

Source Code

The supplementary source code for the project can be cloned at the following url:

https://gitlab.csc.tntech.edu/csc2310-sp21-students/youruserid/youruserid-iteration-01.git

You are being provided all of the stubs for the source code and should implement the four methods listed above. The configuration for compiling and executing the application is similar to that for the lab exam. When creating a run configuration, you should select the `VizSharkJunior` class as the main class.

Testing and Debugging

The `data` directory contains several data files that can be used to test the application. Due to the highly graphical nature of the application, the testing of the code should be conducted by reading CSV data and adding them to the graph, viewing the changes to the graph. The debugger should be your main mechanism for testing the software when you run into issues.

Getting Started

- It is suggested that you go through this description and make a list of all of the elements that you are supposed to implement
- The GraphDemo in-class exercise provides an initial starting point for the project
- The TSharkData class was a part of Lab 04. Use that lab as a reference point for the input data and some initial testing

Submission

Submit your project via git by staging via add, committing the files and pushing to gitlab.

Rubric

- Compilation (15 points)
  - Code fully compiles without error (full credit: 15)

    - Code algorithm(s) generally correct but the program does not compile (partial credit: up to 10 points)
- Execution (15 points)
    - Graph fully displays network data with labels (full credit: 15)
        - Nodes labeled with IP addresses
        - Edges labeled with edge weights
    - Graph partially displays but is not configured correctly (partial credit: up to 10 points)

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 9	Iteration 1	Data source implementation: CSV support
April 23	Iteration 2	Streaming support and animations
April 30	Iteration 3	TBD (Extra Credit)

Iteration 2

In the last iteration, you developed an application that was one-dimensional in that it only supported one type of input: CSV files generated by `tshark`. In this iteration you will be extending the VizSharkJunior application to support other potential data sources, including directly from `tshark` as a live stream of data.

Preliminaries

In order for your application to read data using `tshark`, you must have the Wireshark system installed. Once installed, you should run the application with the following command-line option:

```
tshark -D
```

You should see output similar to the following for Mac or Linux:

1. en0 (Wi-Fi)
2. p2p0
3. awdl0
4. llw0
5. utun0

and the following for Windows:

1. \Device\NPF_{1FD1B9E1-7393-428C-AFEB-922FB574F47B} (Local Area Connection* 3)
2. \Device\NPF_{2CBD17F2-4B0D-4157-81F4-C513DEDA710A} (Local Area Connection* 10)
3. \Device\NPF_{C627A639-7315-427F-A309-BA4077C72FBB} (Local Area Connection* 1)
4. \Device\NPF_{8D8E0761-F65C-4BA6-B69B-F238F569712E} (Wi-Fi 2)
5. \Device\NPF_{CE0A3DC9-CD55-4748-A02B-0FB84CB4826E} (VirtualBox Host-Only Network #2)

The index for the main network interface ("1" for the Wi-Fi interface en0 in the mac case, for instance) will be used in the following command as a replacement for the %INTERFACE% variable shown below:

```
tshark -T fields -e frame.number -e frame.time_relative -e
_ws.col.Protocol -e ip.proto -e ip.src -e ip.dst -e tcp.srcport -e
tcp.dstport -e udp.srcport -e udp.dstport -E header=n -E separator=, -
E quote=d -E occurrence=f -i %INTERFACE%
```

Tasks

In the final required iteration of the class project you will be doing the following:

- You will create an **abstract** class called GeneratorJunior using all of the methods, including the `nextEvents` method, from CSVGenerator as necessary. The GeneratorJunior class should implement the **Generator interface** but should exclude the implementation of the `begin` method, instead specifying it as abstract.
- You will re-implement the CSVGenerator class as a subclass of GeneratorJunior, implementing only the constructor and `begin` methods.
- You will implement a class called TSharkJunior as subclass of GeneratorJunior. It should inherit all of its methods from GeneratorJunior except for the constructor and `begin` methods.

Figure 7 shows the architecture of the intended solution for this iteration. Your implementation will need to address the issues described in the following sections.

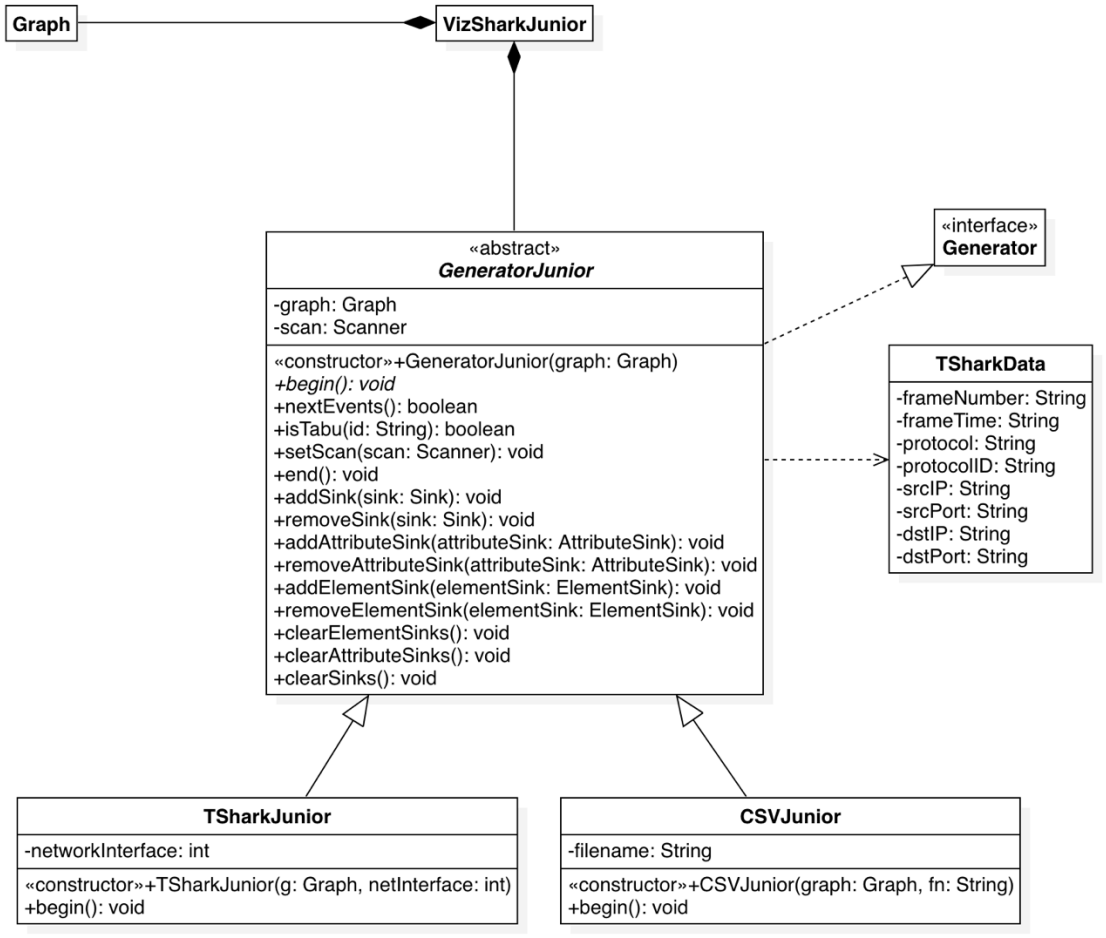


Figure 7 VizSharkJunior Architecture Modified

GeneratorJunior

The GeneratorJunior class is the primary superclass for the data sources supported by the application. The source code for VizSharkJunior shows how the GeneratorJunior class is used, including the use of polymorphism to switch between using either CSV or direct TShark data. Specifically, the code appears as follows:

```
private GeneratorJunior genJr;
…

if (cmd.contentEquals("--csv"))
   this.genJr = new CSVJunior(graph, arg);
else
   this.genJr = new TSharkJunior(graph, Integer.parseInt(arg));
```

The GeneratorJunior class should implement everything found in the CSVJunior class from the previous iteration, while also adding in a getter method called setScan.

CSVJunior

The solution to CSVJunior has been provided to you as the starting point for your refactoring activity. However, the class WILL be fully modified according to the model provided in Figure 7. Your implementation for CSVJunior should remove the explicit implementation of the Generator interface, and only implement the methods shown in the model, including the constructor for saving the graph (by calling the super method) and filename as a String.

TSharkJunior

The TSharkJunior class will implement a constructor and begin method. The begin method should start a tshark process using the following code:

```
Process p = Runtime.getRuntime().exec(cmd);
BufferedReader output = new BufferedReader(new InputStreamReader(p.getInputStream()), 512);
```

Once the process has been started, a scanner object can be created using the output object as follows:

```
Scanner scan = new Scanner(output);
```

The `cmd` method should be created using the command shown in the preliminaries section, making special care to replace the %INTERFACE% placeholder with the input parameter called "netinterface" from the model.

A sample program (`Sample.java`) has been provided that shows how this works and should be used in a manner similar to the begin method from the solution to the CSVJunior class.

Code

The starting point for your solution is provided in the URL listed below. Note that you must make significant changes to CSVJunior and create the GeneratorJunior.java and TSharkJunior.java files. A sample video showing the full operation of the system will be provided.

https://gitlab.csc.tntech.edu/csc2310-sp21-students/yourid/yourid-iteration-02.git

Running the Program

The program is executed using the following command-line arguments:

```
java -cp %CLASSPATH% standalone.VizSharkJunior {--csv filename | --live interface}
```

For example,

```
java -cp %CLASSPATH% standalone.VizSharkJunior --live 4
```

where the network interface is 4, and

```
java -cp %CLASSPATH% standalone.VizSharkJunior --csv data/data2.csv
```

Documentation

In this iteration you will use the markdown notation to document your project, including adding details about how to run the program, library dependencies, etc. Refer to previous iterations for guidance on dependencies. A tutorial on writing markdown documents can be found at

https://gitlab.csc.tntech.edu/csc2310-sp21-students/yourid/yourid-markdown-tutorial.git

Submission

Stage, commit, and push to gitlab as usual.

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 9**~~	~~Iteration 1~~	~~Data source implementation: CSV support~~
April 23	Iteration 2	Streaming support
April 30	Iteration 3	Animations and Layout (Extra Credit)

Iteration 3 (Optional – Up to 10 points extra credit)

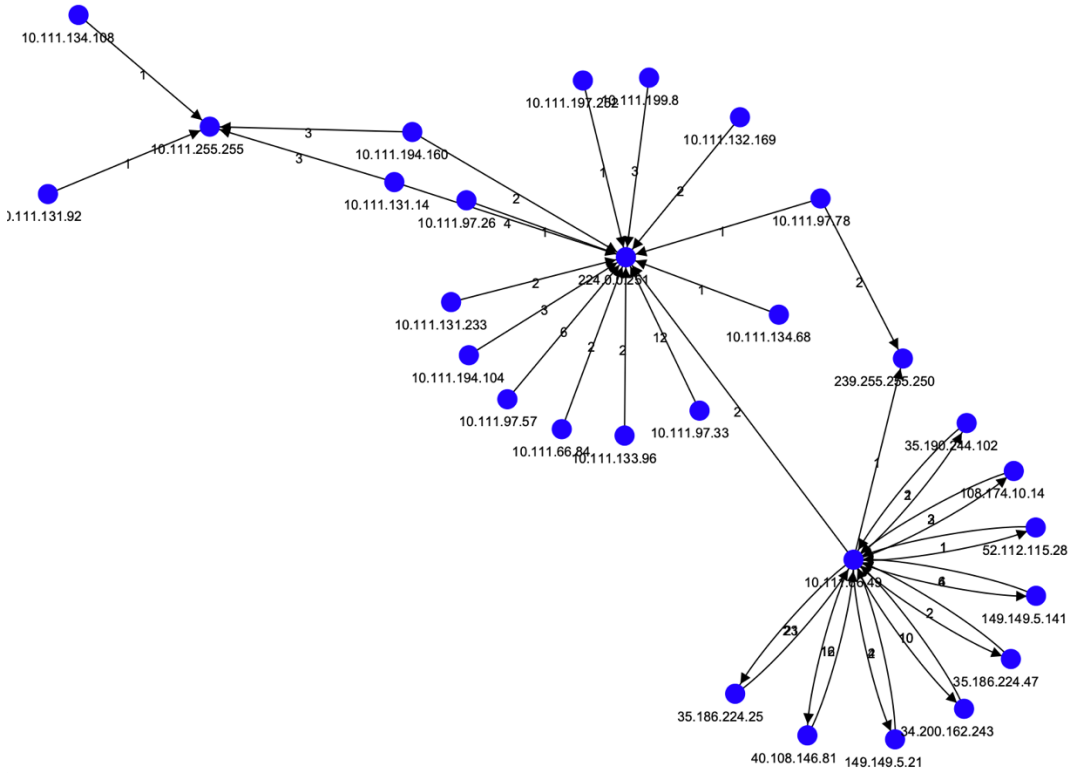
The final (extra credit) iteration of the project will focus on formatting of graphs using GraphStream variant of css.

CSS Formatting and Reference

The GraphStream library supports CSS style specifications for formatting the display of graphs. The current distribution displays graphs using the following css code:

```
node {
    text-alignment: under;
    size: 15px;
    fill-color: blue;
}
```

which produces a graph that appears similar to the following:



The following reference document describes how to format a graph:

https://graphstream-project.org/doc/Advanced-Concepts/GraphStream-CSS-Reference/

The basic structure for formatting elements of a graph is the following:

selector {

    property **:** *setting***;**

}

The available selectors that are most relevant for this exercise are graph, node, and edge. Properties for each of the selectors is specified in the documentation.

To receive credit for this activity, you must make changes to the `graph.css` file as follows:

- Change the node size, and color
- Change the color and size of the text label for nodes
- Change the edge width and color, change the size of the arrow
- Change the color and size of the text label for an edge

Code

The starting point for your solution is provided in the URL listed below.

https://gitlab.csc.tntech.edu/csc2310-sp21-students/yourid/yourid-iteration-03.git

The program in this repository is the same as the initial commit of Iteration 02 and thus can be executed in the same manner.

Submission

Stage, commit, and push to gitlab as usual.

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 9**~~	~~Iteration 1~~	~~Data source implementation: CSV support~~
~~**April 23**~~	~~Iteration 2~~	~~Streaming support~~
April 30	Iteration 3	Layout/Styling (Extra Credit)