Packet Visualization

Packet Visualization System

Developer Document

Version 1.0.1

December 7, 2021

# 

[**1.0 Introduction**](#_8cvy97brhziq) **4**

[1.1 Document Purpose](#_frmo8vasgy33) 4

[1.2 Development Team](#_umb0gnc1c7is) 4

[1.3 Current Working System](#_qkr7fbf6vsk6) 4

[**2.0 System Architecture**](#_5754t1mppm6d) **4**

[2.1 Model View Controller](#_f3fw2agecklp) 4

[**3.0 Database Development**](#_vzzj0kmsze82) **5**

[3.1 MongoDB](#_6pjexa7upc9h) 5

[3.2 MongoManager](#_1mjnktl10uro) 6

[**4.0 Backend Components**](#_hix1h7rijyts) **6**

[4.1 Wireshark](#_fhcvfzscef81) 6

[4.2 Bandwidth Plot](#_r6zghy8vp13o) 6

[4.3 Classifier](#_f0268vuhvra8) 6

[4.4 Controller](#_l7bqagnryfgh) 7

[4.5 Json Parser](#_ybyhirmpsy55) 8

[4.6 Load](#_hclmn2e9v7hc) 8

[4.7 Worker Files](#_7tox5gpzj64j) 8

[4.8 Table Backend](#_kkqciokvluao) 8

[**5.0 Models**](#_gczy9ablfz8f) **9**

[5.1 Dataset](#_fp45gvg2j4x2) 9

[5.2 PCAP](#_xprahgumarrk) 9

[5.3 Pcap\_Worker](#_rpvl0fk9ro1c) 10

[5.4 Project](#_jy9831csolgo) 10

[5.5 Workspace](#_s124fia23nsp) 10

[5.6 Analysis](#_uq27xk2bh74x) 11

[**6.0 UI Components**](#_dv5leh9838h3) **11**

[6.1 Resources (Images and files)](#_jfg4etdhk4f0) 11

[Adding a resource](#_wqq81d6xf3dw) 11

[Updating a resource](#_651i5bn55mq0) 12

[6.2 Filter\_GUI](#_mwrqi4r8oq52) 12

[6.3 Load\_Window/Save\_Window](#_3g725bfvqydh) 12

[6.4 Properties\_GUI](#_qo6jr1lhkj74) 12

[6.5 Properties\_Window](#_41t9j6k6x4jx) 12

[6.6 Startup\_Window](#_x0yyoioloojo) 13

[6.7 Table\_GUI](#_7lgbkfcfdwc8) 13

[6.8 Workspace\_Window](#_5fnu0e4wsktx) 13

[Actions](#_xg1ilrtxpapp) 13

[Menu bar](#_l6sln77x88mu) 14

[Toolbar](#_qtt2v36b1bj0) 14

[Status Bar](#_pls5ewmyoa1d) 14

[Context Menu (Right-click menu)](#_5jn42suwmjqw) 14

[Other](#_93pi3dl5thyx) 14

[**7.0 Pipeline/Configuration**](#_7epn34t1fa0n) **14**

[7.1 Testing](#_xmngr2gakagd) 14

[7.2 Deployment](#_icc6gctkbkr) 15

[**8.0 Pending Work**](#_n8000jmjelnm) **15**

# 

# 1.0 Introduction

## 1.1 Document Purpose

The purpose of this document is to inform any future development teams of the currently developed system including the architecture of the system, key models, database development, and full configuration. The document provides insight to development features as they currently exist and considerations for future development.

## 1.2 Development Team

Version 1.0 of the system was developed by the following development team:

Adrian Belmontes

Eyan Meraz

Alex Vasquez

Timmy Williams

Abraham Lomely

## 1.3 Current Working System

At handoff to the client, the Packet Visualization system fulfills all requirements that are stated in the original requirement document. Current pending items are related to usability features that are not explicitly stated in the document and performance features that enhance system computation speed and usability.

# 2.0 System Architecture

## 2.1 Model View Controller

The system is developed with a model view controller architecture, where the models can be found in the models directory. These models provide core functionality to the system and will be discussed in detail in section 5.

The application core is broken down in 3 main folders: ui\_components, models and backend\_components.

Ui\_components: This folder hosts everything that the application is using for the GUI, here you will be able to find everything that is displayed in the front end.

Models: This folder holds the main entities that our system works around, the entities can be found in the class form and they dictate, with the exception of packets, the shape of each entity that our system uses in order to function. The folder also contains entity operator which is the class that every component should be using in order to interact with the MongoDB.

In the system there are instances where models require some controller to communicate with the front end of the system and there are instances where the front end can directly communicate with the system depending on what functionality was needed with the given model. These will be discussed in detail in a later section.

Backend\_components: This folder contains the business logic of our system and data transformation modules.

Tech Debt: The pattern is not perfect since we were learning still, so certain modules may not be obeying the rules we are describing in here. If you find any modules that need to be refactored keep in mind that there is known tech debt that due to the timeline of our project was not possible to fix, so feel free to modify and create a pull request for it.

# 3.0 Database Development

## 3.1 MongoDB

The database of choice for the Packet Visualization system is MongoDB. MongoDB provided the development team with a flexible way to insert packet data while maintaining a hands off approach to .pcap file extensions.

.pcap files are handles in the following fashion:

The packet visualization system determines whether the pcap file is a large pcap file or a small pcap file (some threshold that the developer can decide). When a large pcap file is ingested, the system leverages editcap technology to split the large pcap into smaller chunks. This helps us avoid any memory issues when processing packet data in the pcap to json format. When dealing with a small pcap, the system leverages tshark to transform the pcap into JSON. The JSON is then read into the database (Approx 50,000 packets/10sec).

Upon successful insert, the system will only reference database data for computation, however original pcap files will be kept in the dataset directory for other use cases. All files used for processing and database insert are cleaned up and removed from the directory created by the system.

## 3.2 MongoManager

File: packetvisualization/models/backend\_components/mongo\_manager.py

The MongoManager file can be found in the backend\_components directory. The MongoManager is responsible for handling any interaction with the database. This includes creating a connection to the database, creating a new database for every new dataset that exists in the project, inserting and removing data, dumping a database (essentially saving) into the workspace directory, and restoring a database for a workspace that was previously saved.

It is important to note that in the front end of the system there is a reference to the database name, and queries are not strictly written in this file. *This was a major consideration from the development team and may be something to consider as far as moving any function that queries that database to this location.*

# 4.0 Backend Components

The following section will give a brief description of the files Backend Component directory along with uses.These files are located in the packetvisualization/backend\_components directory. Note: The worker files will fall under section 4.7 as they all perform a similar task for the system.

## 4.1 Wireshark

File: Wireshark.py

The Wireshark file has functions that make use of wireshark features. The functions in this file allow the user to open wireshark and apply wireshark filters on a given pcap. These functions are intended to be used from the front end whenever a user selects some wireshark related action.

## 4.2 Bandwidth Plot

File: bandwidth\_plot.py

The Bandwidth\_plot files responsibility is to create the Time vs. Bandwidth graph that is seen in the front end. This file makes use of the plotly package and X and Y coordinates that are pulled from packets in the database.

## 4.3 Classifier

File: classifier.py

The classifier class is a very simple scikit-learn implementation of the k-means algorithm.

In order to run the classifier you will need to create a new instance fulfilling two parameters: cluster\_number (self explanatory) and *context\_results*.

Context\_results: This parameter expects mongo results, as long as the data has similar shape, pandas will be able to create a dataframe out of it.

Cluster\_number: It is an integer that dictates the clusters (groups) that k-means needs to create. As a suggestion, do not test with 1 cluster or 1 per packet cluster, the typical use case is to be able to identify outliers in the dataset.

The Classifier is composed of 2 stages of data preprocessing and classifying by k-means. The first stage transforms mongo DB data to a DataFrame using the Python Pandas library, and then it transforms any categorical value to numeric to create classifiable features. The second stage runs the classifier with the pandas DataFrame and the cluster number defined by the user, which then populates the following properties:

Feature\_list: A python string array that stores the column names of the numerical information.

Centroids: Once the algorithm runs, it will create an X amount of centroid vectors that is equal to the number of clusters expected.

Result\_data\_frame: The result DataFrame is the classified data, back in categorical, with the instance number column and cluster number column attached to each packet data.

The instance number column is the temporary unique identifier for each packet to be displayed in a plot.

The cluster number is the cluster the packet belongs to once it is classified

## 4.4 Controller

File: controller.py

The controller class objective is to be the API-like class that would open interaction between front-end and back-end components, in the end the functionality placed in here is only to run the classifier algorithm. This controller currently holds only one functional endpoint which is:

Create\_analysis: This method is an endpoint for any GUI application to be able to run the classifier.

Inputs:

Uuid\_list: The mongo ID list of the selected packets to classify

Properties\_list: The user selected properties to classify the data

Cluster\_number: Number of “groups of data required”

Obj: Dataset name

Db: Database name

Output:

Classified\_data: DataFrame with the already classified data

Classifier.feature\_list: Features or properties that were classified (for plotting purposes)

## 4.5 Json Parser

File: json\_parser.py

The Json\_Parser file plays a significant role in the data that is received from a given query. Given that the json is multi nested, the development team had to find a way to select individual fields for given use cases. The functions in this file solve this issue and allow developers to easily reference a field in the nested json.

## 4.6 Load

File: load.py

The Load file is responsible for setting up the system in the case that the user selects “Load Workspace” and provides a zipped workspace as user input. The directory that is saved when a user saves a workspace contains save files that allow the system to parse and reload data in a new instance of the Packet Visualization client.

## 4.7 Worker Files

Fille: load\_worker.py, plot\_worker.py, save\_worker.py

Worker files in this directory are set up to be processed by a thread in the front end. They allow the system to process some functions and perform computation without freezing the application. For any future heavy computation use cases, these files can serve as a template for when the need for a separate thread. They include signals that are sent to the main GUI interface such as progress reports, data, or finished a finished signal to indicate the thread completed its work.

## 4.8 Table Backend

File: table\_backend.py

There is a use case for the system that requires the user to be able to see individual packets from a given dataset or pcap file. This file introduces the logic to be able to view individual packets from the database.

There is also a use case that requires the user to be able to select these packets and export them to various types such as ASCII, JSON, and another PCAP. The functionality for those export types is found in this file.

# 5.0 Models

The following section details the main models used in the system. Each file contains similar code with different details depending on required use cases. These files are located in the packetvisualization/models directory.

Each of the models in this section have the following functions:

Save: Allows us to wrap up any object of a given type along with associated information and add it to a save file.

Remove: Allows us to remove a given object from the system in its entirety. (Both the workspace directory and the instance of the object)

Delete: Provides the inner workings of the Remove and deletes all information related to the object.

## 5.1 Dataset

File: dataset.py

The Dataset model contains all information related to a dataset. It is important to note that a major data structure that lives inside of a Dataset is a list of PCAP objects. This allows use to keep track of which PCAP objects belong to a given dataset.

The Dataset class is responsible for handling the creation of its own directories within the system. Upon creating a dataset, a new directory with the dataset name will appear within the workspace directory.

See user manual “Add Dataset” for more information.

## 5.2 PCAP

File: pcap.py

The PCAP model contains all information related to a PCAP. It is important to note that upon initialization of a PCAP object there is a threshold that is set by the system to determine whether to treat the small PCAP or a large PCAP. In the instance of a Large PCAP the system will start to split the PCAP into smaller chunks for processing purposes.

This file will contain all the functions to process .pcap data into .json format (necessary for DB insert). The file will also contain the cleanup associated after data is successfully inserted into the DB. This includes the directory where the split pcaps sit and the directory where the associated JSON for each splitcap sits. This was a necessary move to limit the size of the workspace directory as .json file extensions often take up more memory than the development team would like.

## 5.3 Pcap\_Worker

File: pcap\_worker.py

The Pcap\_Worker files contain the function to process .pcap file extension data and insert it into the database. Inside of the run function, you will find the functionality which takes a directory for split caps associated with a large pcap and transforms them into json (tshark command used). Once this process is complete the system will then work on adding each of the json files data into the proper database.

The reasoning for this relates to limited processing power. Typically in this case, we would rely on a data processing engine to handle large sums of data and perform computation; however for the allotted resources the development team resorted to handing off this process to another thread to work in the background. There is also an associated progress bar that keeps track of the progress for these functions and displays them on the front end.

## 5.4 Project

File: project.py

The Project file contains all information related to a project. In similar fashion, the project stores a list of Dataset objects, this is necessary to keep track of the Datasets associated with a given project. This is also extended to an analysis as the development team considered an analysis to be on the same level of the system hierarchy.

There are also various functions in the project class that help provide information on the class such as the overall project size. These are required to fulfill various use cases related to displaying information on a given project.

## 5.5 Workspace

File: workspace.py

The Workspace file contains all information related to a Workspace. In similar fashion, the workspace stores a list of Project objects, this is necessary to keep track of the Project associated with a given Workspace. The workspace class will handle creating a directory for a given workspace, and will hold nested directories and files that include all of the models listed in the section.

The save function in this file will handle creating the actual ZIP in the instance that a user would like to save their work.

## 5.6 Analysis

File: analysis.py

The Analysis class contains all information related to an Analysis. Currently this file provides no functionality besides the functions described in the intro of the section, it is only used to store information for the analysis properties and to save that information to the save file

# 6.0 UI Components

The ui components are the files that handle what the user will be seeing when interacting with the system. These files are located in the project folder under packetvisualization/ui\_components.

## 6.1 Resources (Images and files)

Files: resources/qrc\_resources.py, resources/resources.qrc

The resource file that the project uses to access icons and files is located in the packetvisualization/ui\_components/resources folder. The “resources.qrc” file holds the name of the resource that will be accessed along with the path to the file, relative to the resources folder.

### Adding a resource

If a new picture wants to be added, the “resources.qrc” file will need to be updated as well as running the following command: “pyrcc5 -o qrc\_resources.py resources.qrc”

This command will update the qrc\_resources.py file which is the resource file used by our project.

### Updating a resource

To update a resource such as including an updated version of the user documentation to the package, all the developer would have to do is run the following command, make sure to keep the name of the file the same: “pyrcc5 -o qrc\_resources.py resources.qrc”

## 6.2 Filter GUI

File: filter\_gui.py

This gui is used when the user selects to filter some packets. It allows the user to create a new dataset based on the filtered packets. The filter requires the same syntax as wireshark would use for their system.

## 6.3 Load Window/Save Window

File: load\_worker.py, save\_worker.py

The load and save windows are a simple loading bar, a label to give a bit more description and an “OK” button that closes the window. The progress and status description is updated on the workspace\_window, which receives updates from the threads handling the loading of a project and saving of a project, respectively.

## 6.4 Properties GUI

File: properties\_gui.py

Properties GUI is the window that is in charge of running the classification methodolody, once packets are selected to be analyzed from the table GUI, it is composed by three sections: properties, algorithm chooser and cluster number selection.

Once the user selects an algorithm (k-means for the current scope of the system), selects the properties to classify and the number of clusters to run, then the analyze button runs the analysis method which is the one that calls the classifier via a controller and then receives back the classified data.

When the classifier is done running, it prompts the user with an options window to name the analysis and then it closes the current properties GUI to open the scatter plot with the classified DataFrame from the analysis.

## 6.5 Properties Window

File: properties\_window.py

This window is used to display the details pertaining to a project, dataset, and an analysis. When one of these objects is right-clicked on the project tree, the user is given the option to view properties of that object. If a new type of object wants to be added, the type would need to be added to the get\_properties() method and create a new method to populate the window. This would also have to be added to the context menu on the workspace\_window to allow the option to pop up when it is right-clicked.

## 6.6 Startup Window

File: startup\_window.py

The startup window is used, as the name suggests, as the starting window when the system is launched. This window allows the user to select between creating a new workspace or loading an existing save file (zip).

## 6.7 Table GUI

File: table\_gui.py

This file contains everything related to the table created from the dataset information. This gui includes everything from the fields displayed to the options displayed when the user right-clicks on components of the table as well as all the methods performed for those actions. Everything related to the table can be modified here.

## 6.8 Workspace Window

File: workspace\_window.py

This is the main window that connects all the backend components with the front-end that is displayed to the user. At initialization, the program is designed to create all the windows that will be used by the user. If it is loading an existing workspace, an additional step is taken to create and load all backend components including importing data into the database. Below will be a description on what actions are and how you can add new actions to the workspace.

### Actions

The \_create\_actions() method will contain all actions that are possible in the system, these are the links between a gui component, such as a button, to a method. These actions can be customized to have a name, icon, a shortcut that the user can use and a custom message that is displayed to the user on the status bar or tooltip. After creation, the actions are connected to methods in the \_connect\_ actions() method.

### Menu bar

Our system has a menu bar at the top, you are able to add the actions that you just created to the menu in the \_create\_menu\_bar() method.

### Toolbar

As a user gets more experience with the system, they may request to add the most used actions to the toolbar that originally is located on the left. To add actions here, you can add them in the \_create\_tool\_bar() method.

### Context Menu (Right-click menu)

The context menu can be customized in the ContextMenuEvent() method. Here you can set conditions on which actions you want to display when an object is right-clicked.

### Status Bar

The status bar is initialized at \_create\_status\_bar(). The look and customization of this bar can be changed in this method.

### Other

The other main methods to describe are the thread methods which are used to handle operations in a new thread to avoid freezing the GUI, reporting methods that communicate with the threaded methods to update GUI components such as progress bars, and all other methods that handle operations in our system. All methods are named appropriately and have a brief description of what they do.

# 7.0 Pipeline/Configuration

The system is hosted in a GitLab repository, runs through a CI/CD pipeline dictated by yaml file instructions (found in code). It runs 2 jobs: test and deploy.

We are currently hosting the project in a free-tier which means that it has some limitations that can be found here: <https://about.gitlab.com/pricing/>

## 7.1 Testing

Out testing is done using the pytest tool. The testing files are located in the directory tests/. Creating new tests is as simple as adding the testing methods to already existing files or adding new files. Tests files do not need to be numbered but we numbered them so that pytest will execute the files in that order; so that a workspace will be tested before the project is tested. Locally, these tests can be run by simply installing pytest and running the command “pytest”. If a specific test wants to be run, the command to run would be “pytest <path to file>”.

In addition to testing locally, the test’s are programmed to run automatically in our repository by our pipeline when a developer pushes to any branch. This assures the system is running properly before merging to other branches.

## 7.2 Deployment

The pipeline deploys to a Pypi package which is called packetvisualization and instructions on how to run or install can be found in our readme.md file.This deployment process is also automated to run once the test’s all pass when there is a merge or push to the main branch only.

If new python dependencies were installed, add them to the ‘startup.py’ file so that they are installed when the package is installed.

# 8.0 Pending Work

Currently the system meets the requirements found in the first requirements document. The only pending work involved further testing, the coverage is limited due to time constraints however the template and working testing files currently exist in the testing directory.

There are also enhancements that can be made to the systems performance by introducing threads that handle other operations. Currently threads only handle operations that require heavy computation however this can be furthered for any and every action.

There are various files that still need full testing, due to time constraints the development team was not able to provide a full testing capability for every file in the system.

Lastly,the load bar of the system does not display a counter with a number, rather just a bar that increments as computation completes. The next step for this load bar is to add a visual number counter to give the user more of a sense of how long the computation is going to take.