POLITECNICO
MILANO 1863

Wi-Fi encrypted traffic classification

Project of the Wireless Internet course

Author
Christian Confalonieri

Professor
Alessandro Redondi

2023/2024

Contents

1 Introduction

The primary objective of this project is to design and implement a machine-learning classifier that can distinguish between different types of user activities based on Wi-Fi traffic data, without decrypting the packets. The system will be trained on a dataset collected using Wireshark and will also be able to classify "real-time" traffic data.
The following key operations will be performed:

- **Traffic Sniffing**: data for training and testing the classifier will be collected using Wireshark. This tool will capture packets from a Wi-Fi network, and the data will be saved in a .csv file.
  For real-time traffic analysis, the data will be collected using the pyshark library, a Python wrapper for tshark, the command-line version of Wireshark.
  More details about the data collection process will be provided in the next sections.
- **Feature Extraction**: once the traffic data is captured, we will extract various statistical features at regular intervals (every W seconds). These features will include:
  - Number of uplink and downlink packets: the number of packets sent and received by the device.
  - Packet size statistics: average, variance, and maximum packet size (minimum size is not considered since it is always 60).
  - Inter-Arrival Times: average and variance of the time intervals between packets.

  Other features, such as sequence number and packet type/subtype statistics, can also be considered. However, for this project, the above features are sufficient to achieve good classification accuracy.
- **Training and Evaluation**: using the extracted features, we will train a machine-learning classifier to categorize user activity. The classifier will differentiate among activities such as idle, web browsing, Spotify, and YouTube streaming.
- **Performance Evaluation**: the accuracy of the classifier will be assessed through a confusion matrix. This matrix will help evaluate how well the system classifies different user activities and identify any areas for improvement.
  In addition, a function, `test_window_size()`, will be implemented to test the classifier with different window durations and plot the results to find the best one.
- **Real-time Traffic Analysis**: the classifier will be used to analyze real-time traffic data. The system will continuously monitor the network and classify user activities as they occur. Data will be analyzed every W seconds, and the results will be displayed on the console.

2 Considerations and Assumptions

2.1 Operating System

This project was developed and tested on a Linux machine, specifically using the *Fedora 40* distribution. The code should also work on other operating systems, but some modifications may be needed to adapt it to different environments.

2.2 Sudo Permissions

The training and evaluation part does not require special permissions. However, the real-time traffic analysis part does. To capture packets from a network interface, the script must be run with root privileges.

2.3 Data Collection

As previously mentioned, the data used for training and testing the classifier will be collected using Wireshark. For real-time traffic analysis, the data will be collected using the pyshark library. Both methods require root privileges to capture packets from a network interface.

2.3.1 Wireshark

Wireshark must be configured to display the following columns:

- **No.**: packet number.
- **Time**: time of the packet, starting from 0.
- **Transmitter**: MAC address of the transmitter → *wlan.ta*
- **Source**: MAC address of the source → *wlan.sa*
- **Receiver**: MAC address of the receiver → *wlan.ra*
- **Destination**: MAC address of the destination → *wlan.da*
- **Length**: length of the packet.
- **Sequence Number**: sequence number of the packet → *wlan.seq*
- **Type/Subtype**: type/subtype of the packet → *wlan.fc.type_subtype*

It is important to have this exact configuration; otherwise, the script will not work properly.
Data must be saved in a .csv file, following the naming structure: *<activity>_<whatever>.csv*, where *<whatever>* is a string that can be used to differentiate between different captures of the same activity.
The file must be saved in the *test_data* folder, which must be located in the same directory as the script.

2.3.2 Wi-Fi Band: 2.4 GHz

The data collection process for the test files provided and used in this project was carried out on the 2.4 GHz band. Initially, the router was capable of operating on both bands, but the 5 GHz band was disabled during the data collection phase. This was done to simplify the analysis and focus on a single band, as the target device could otherwise switch between the two bands, complicating the analysis.
When switching to monitor mode, the network interface must be set to the same band as the target device; otherwise, packets will not be captured. The channel used for testing was channel 11, a commonly used channel in the 2.4 GHz band.
For more info on Wi-Fi channels, refer to: `https://en.wikipedia.org/wiki/List_of_WLAN_channels`

2.3.3 Monitor Mode

To capture Wi-Fi traffic data, the network interface must be set to monitor mode.
A simple way to do this is by using the *airmon-ng* tool, which is part of the *aircrack-ng* suite.
The following command can be used to enable monitor mode on a network interface:

```
sudo airmon-ng start <interface> <channel>
```

where *<interface>* is the name of the network interface (you can use the *iw dev* command to list all available interfaces) and *<channel>* is the Wi-Fi channel to monitor. The command will create a new interface in monitor mode, usually named *<interface>mon*.
To stop monitor mode, you can use the following command:

```
sudo airmon-ng stop <interface>mon <channel>
```

where *<interface>mon* is the name of the interface in monitor mode, and *<channel>* is the Wi-Fi channel to set the interface back to.
In both cases, you can omit the channel parameter if you want to automatically select the channel, but you should specify it if your router is dual-band (set it to the same band as the target device).

2.4 Features and Activities

2.4.1 Features

The features extracted from the traffic data are:

- **Number of uplink packets**: the number of packets with the source equal to the device MAC.
- **Number of downlink packets**: the number of packets with the destination equal to the device MAC.
- **Average packet size**: the average size of the packets.
- **Variance of packet size**: the variance of the packet size.
- **Maximum packet size**: the maximum size of the packets.
- **Average inter-arrival time**: the average time between packets.
- **Variance of inter-arrival time**: the variance of the time between packets.

In the script, data is filtered to consider only packets with the device MAC address as the transmitter, source, receiver, or destination. This ensures that only packets sent or received by the device are considered, excluding packets simply passing through the network.
As mentioned before, other features can be considered as well, but for this project, the above features are sufficient to achieve good classification accuracy. The minimum packet size is not considered since it is always 60 bytes.

2.4.2 Activities

The activities that the classifier can distinguish are:

- **Idle**: the device is connected to the network but is not performing any activity (screen is off).
- **Web browsing**: the device is browsing the web. This activity is heavily influenced by the user's behavior, such as opening multiple tabs, scrolling through pages, clicking on links, and visiting websites with lots of images or videos.
- **Spotify**: the device is streaming music from Spotify. The user can interact with the app, for example, by changing the song, adjusting the volume, or having the screen off.
- **YouTube**: the device is streaming a video from YouTube. The user can interact with the app, such as changing the video, skipping parts of it, or having no interaction at all.

Activities are defined based on user behavior and the type of traffic generated by the device. The classifier will be trained to recognize these activities based on the features extracted from the traffic. More activities can be added by collecting additional data, as specified in the previous section.

3 Implementation in Python

3.1 Libraries and Dependencies

The two scripts are written in Python and require the following libraries:

- **json**: to read the configuration file.
- **os**: to manage files and directories.
- **pandas**: to read the .csv files.
- **matplotlib.pyplot**: to plot the results.
- **numpy**: to perform mathematical operations.
- **subprocess**: to run shell commands.

- **sklearn.preprocessing**:
  - **LabelEncoder**: to generate the encoder for the labels.
- **sklearn.ensemble**:
  - **RandomForestClassifier**: to generate the classifier.
- **sklearn.model_selection**:
  - **train_test_split**: to split the data into training and testing sets.
- **sklearn.metrics**:
  - **accuracy_score**: to calculate the accuracy of the classifier.
  - **classification_report**: to generate a classification report.
  - **confusion_matrix**: to generate a confusion matrix.
  - **ConfusionMatrixDisplay**: to display the confusion matrix.
- **pyshark**: to capture packets in real-time.
- **time**: to manage time intervals.
- **csv**: to write .csv files.

As you can see, libraries such as `subprocess` are used to run shell commands, which is why I cannot guarantee that the script will work on all operating systems. The script was tested on a Linux machine, and it should work on other Unix-based systems as well.
You can install the required libraries using the following command:

```
pip install pandas matplotlib numpy scikit-learn pyshark
```

It is also important to note that the real-time traffic analysis script utilizes the training and evaluation script to classify the traffic. This approach avoids code duplication and keeps the implementation simple.

3.2 Training and Evaluation

This is the first part of the project, where the classifier is trained and evaluated on the test data. As mentioned before, it does not require any special permissions to run.

3.2.1 General Behavior and `main()` Function

The script is straightforward: it reads the configuration file, loads the data, extracts features, trains the classifier, evaluates it, and plots the results.

3.2.1.1 Configuration File

The configuration file is a .json file that contains the following parameters:

- **window_size**: the duration of the window in seconds.
- **target**: the target device MAC address alias, defined in the next parameter.
- **known_mac_addresses**: a dictionary that contains the known MAC addresses and their aliases. The target device MAC address must be present in this dictionary.

Users can modify these parameters to adapt the script to different scenarios.
To add a new alias, simply add a new key-value pair to the dictionary with the following structure:

```
"<alias>": "<MAC address>"
```

Naming known MAC addresses is especially useful in real-time traffic analysis, as it makes the traffic display on the console easier to read compared to using raw MAC addresses.
The configuration file *config.json* must be saved in the same directory as the script.

3.2.1.2 Main Function

At the beginning of the script, the *__pycache__* folder is removed if it exists. This folder is created automatically when the script is started and is normally removed when the script is closed. However, if the script is run with root privileges, the folder will be created with root permissions, and the script will not be able to remove it upon closure.
Since I have not found a way to avoid this behavior, I have added a line of code to remove the folder at the start of the script:

```
if os.path.exists("__pycache__"):
    subprocess.run(["rm", "-r", "__pycache__"])
```

This is not an ideal solution, but it works.
The script then reads the configuration file and waits for the user to press a key to start the training and evaluation process.
The function *analyze_data()* is called next, which reads the data, extracts the activities and features, and returns them (details are provided in the next sections).
We then need to encode the activities, so a label encoder is created and used to encode the activities.
The data is split into training and testing sets, and the classifier is trained on the training set.
To keep the implementation simple, specific parameters in the *train_test_split()* function are not used. Next is the code snippet used to split the data:

```
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(features, encoded_activities,
     test_size=test_size, shuffle=True)
```

The test size is set to 0.2, meaning that 20% of the data is used for testing and 80% for training.
Shuffle is set to True, which means that the feature windows are shuffled before splitting to avoid any bias.
Parameters can be tuned using functions like *GridSearchCV()* or *RandomizedSearchCV()*, but this is not necessary for this project since the classifier already achieves good results.
The classifier is then evaluated on the testing set, and the results are printed on the console.
Finally, the confusion matrix is generated and displayed using the *generate_confusion_matrix()* function.
The main function returns the accuracy, the classifier, and the encoder so they can be used in the real-time traffic analysis script.
Optionally, you can test the classifier with windows of different durations using the *test_window_size()* function, which plots the results to find the best window size. This functionality is commented out in the script but can be easily enabled by uncommenting the function call.
Note: The *test_window_size()* function requires some time to run as it tests the classifier with different window sizes and plots the results. During this time, the script will print each tested window size, allowing the user to track the progress.

3.2.2 Auxiliary Functions

3.2.2.1 `read_config`

This function reads the configuration file and returns the parameters.
In *config.json*, the user can change the window size, add or remove known MAC addresses, and select the target device from the list of known devices.

3.2.2.2 `resolve_mac_address`

This function takes a MAC address and returns the corresponding alias if it is present in the known MAC addresses dictionary. If the MAC address is not found, the function returns the MAC address itself.

3.2.2.3 `print_main_info`

This function prints key information about the script, such as the window size and the target device's MAC address and alias. It also prints a subtitle to help separate different parts of the script (e.g., distinguishing the training and evaluation phase from the real-time traffic analysis phase).

3.2.2.4 `return_csv_files`

This function returns the names of .csv files in the specified directory. For this part, it is used to read data files from the *test_data* folder.

3.2.2.5 `generate_confusion_matrix`

This function generates the confusion matrix, saves it as a .png file, and displays it in a separate window. The generated image can be found in the *results* folder.

3.2.2.6 `filter_data`

This function filters the data to include only packets where the device MAC address is either the transmitter, source, receiver, or destination. This ensures that only packets sent or received by the device are considered, excluding packets that are merely passing through the network.

3.2.2.7 `extract_features`

This function extracts features from the data. The features extracted include the number of uplink and downlink packets, the average, variance, and maximum packet size, and the average and variance of the inter-arrival times. Details about these features are provided in the previous sections.
NaN values are handled by replacing them with 0 or -1, depending on the feature.

3.2.2.8 `analyze_data`

This critical function reads the data, extracts activities and features, and returns them.
Within this function, there is an auxiliary function called *split_data* that divides the data into windows of the specified duration. This function returns a list of windows, each containing data for a specific time window. Specifically, it reads the .csv files from the *test_data* folder, processes each file to split the data into windows of the specified duration, filters the data, extracts features and activities, and appends them to corresponding lists. It also prints the progress as it processes the files. Once all files are processed, the function returns two lists: one containing lists of feature windows and another containing the corresponding activities (the lists are ordered consistently, with each element in the features list corresponding to the same index in the activities list).
For each window, some columns are dropped if they are not used in the feature extraction process. This optimization improves code efficiency.

3.2.2.9 `test_window_size`

This function tests the classifier with windows of varying durations and plots the results on a graph. The function tests window sizes ranging from 1 to 120 seconds, though this range can be adjusted by modifying the function. The *train_test_split* function uses the same parameters as before, except the test size, which can be adjusted by the user (as commented in the *main* function, it is set to the same value as before).
The function prints the accuracy of the classifier for each tested window size and plots the results on a graph. The graph is saved twice: once in the *results* folder, overwriting any previous file, and once in the *results/accuracy_vs_window_old_tests/* folder, where the filename structure is:

```
accuracy_vs_window_<pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')>.png
```

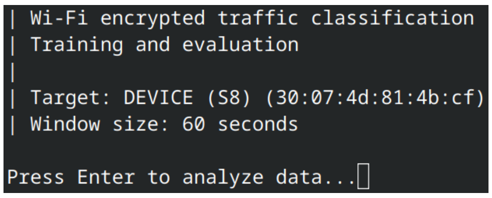
This dual saving helps keep track of previous tests for comparison.
Note: As mentioned earlier, this function requires some time to run because it tests the classifier with different window sizes and plots the results. During this time, the script prints each window size tested to keep the user informed of the progress.

3.2.3 Execution Demonstration

The script is straightforward to run. Simply execute it using Python 3:

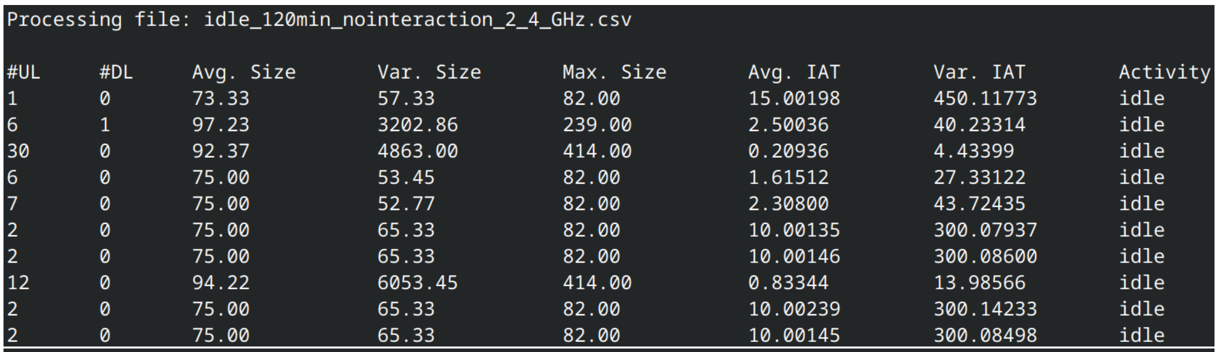
```
python3 training_and_evaluation.py
```

Upon execution, the script will print the main information, read the configuration file, and wait for the user to press Enter to start the training and evaluation process:

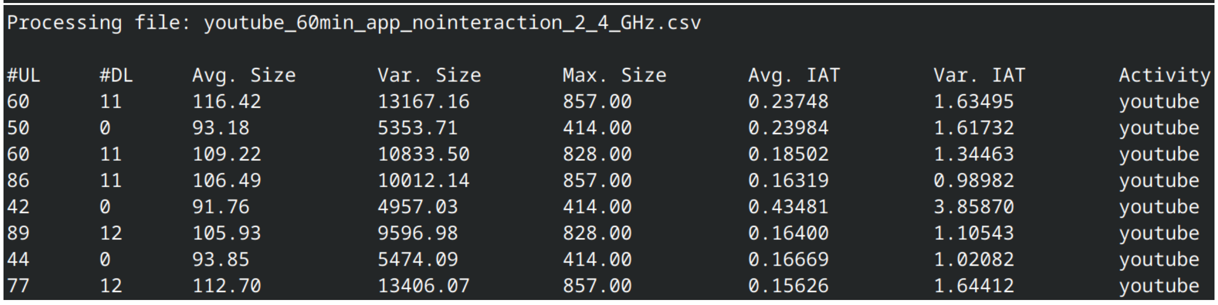


```
| Wi-Fi encrypted traffic classification
| Training and evaluation
|
| Target: DEVICE (S8) (30:07:4d:81:4b:cf)
| Window size: 60 seconds

Press Enter to analyze data...
```

The script will then proceed to read the data, split it into windows, and extract the features and activities. Below are examples of the progress printed on the console:

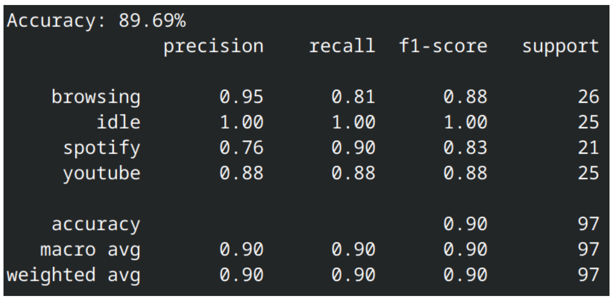


```
Processing file: idle_120min_nointeraction_2_4_GHz.csv

#UL     #DL     Avg. Size       Var. Size       Max. Size       Avg. IAT        Var. IAT        Activity
1       0       73.33           57.33           82.00           15.00198        450.11773       idle
6       1       97.23           3202.86         239.00          2.50036         40.23314        idle
30      0       92.37           4863.00         414.00          0.20936         4.43399         idle
6       0       75.00           53.45           82.00           1.61512         27.33122        idle
7       0       75.00           52.77           82.00           2.30800         43.72435        idle
2       0       75.00           65.33           82.00           10.00135        300.07937       idle
2       0       75.00           65.33           82.00           10.00146        300.08600       idle
12      0       94.22           6053.45         414.00          0.83344         13.98566        idle
2       0       75.00           65.33           82.00           10.00239        300.14233       idle
2       0       75.00           65.33           82.00           10.00145        300.08498       idle
```



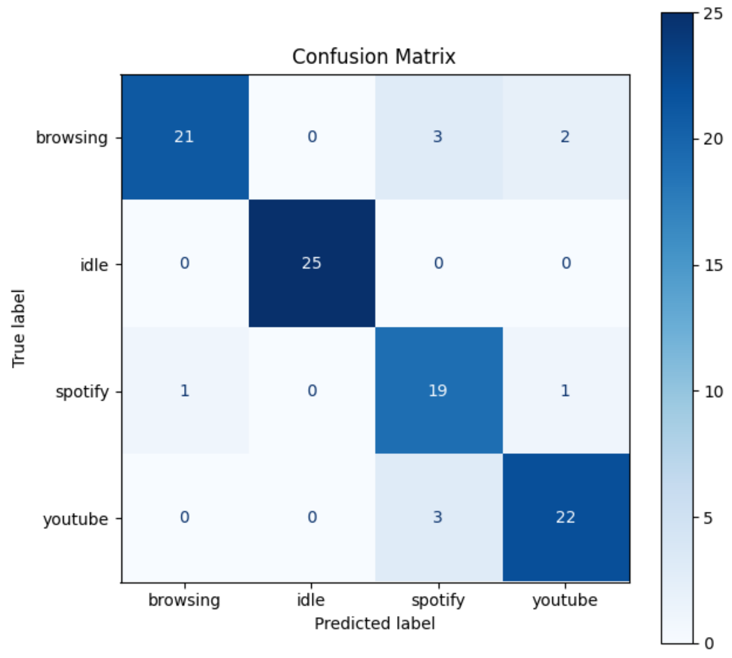
```
Processing file: youtube_60min_app_nointeraction_2_4_GHz.csv

#UL     #DL     Avg. Size       Var. Size       Max. Size       Avg. IAT        Var. IAT        Activity
60      11      116.42          13167.16        857.00          0.23748         1.63495         youtube
50      0       93.18           5353.71         414.00          0.23984         1.61732         youtube
60      11      109.22          10833.50        828.00          0.18502         1.34463         youtube
86      11      106.49          10012.14        857.00          0.16319         0.98982         youtube
42      0       91.76           4957.03         414.00          0.43481         3.85870         youtube
89      12      105.93          9596.98         828.00          0.16400         1.10543         youtube
44      0       93.85           5474.09         414.00          0.16669         1.02082         youtube
77      12      112.70          13406.07        857.00          0.15626         1.64412         youtube
```

After processing, the classifier is trained and evaluated. The results are then printed on the console:



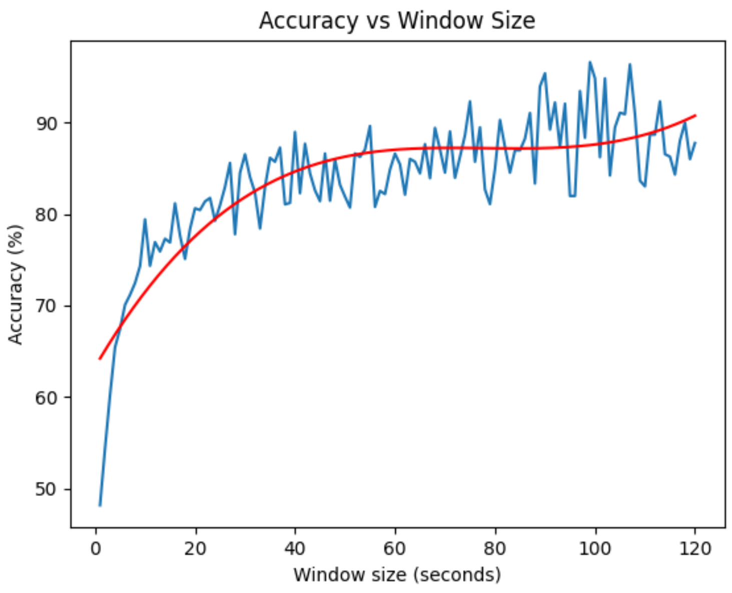
```
Accuracy: 89.69%
              precision    recall  f1-score   support

    browsing       0.95      0.81      0.88        26
        idle       1.00      1.00      1.00        25
     spotify       0.76      0.90      0.83        21
     youtube       0.88      0.88      0.88        25

    accuracy                           0.90        97
   macro avg       0.90      0.90      0.90        97
weighted avg       0.90      0.90      0.90        97
```

Additionally, the confusion matrix is computed, displayed in a separate window, and saved in the *results* folder:



3.2.3.1 Test Window Size

The user can uncomment the *test_window_size()* function call to test the classifier with windows of various durations and plot the results. This function will print the accuracy of the classifier for each window size tested and generate a graph:



As illustrated, the graph appears to stabilize around 60 seconds, suggesting that this is likely the optimal window size. However, window sizes smaller than 60 seconds may also be considered, as the accuracy remains high.

3.3 Real-time Traffic Analysis

This section covers the second part of the project, where the classifier analyzes real-time traffic data. As noted earlier, root privileges are required to run this.

3.3.1 General Behavior and `main()` Function

This script uses a training and evaluation script to classify traffic. The main function is straightforward: it reads the configuration file, loads the classifier and encoder, waits for the user to select a network interface, checks if the interface is in monitor mode, and then starts the real-time traffic analysis.

3.3.1.1 Main Function

The script reads the configuration file and loads the classifier and encoder using the training and evaluation script.
Next, the user is prompted to select a network interface, and the script verifies if the interface is in monitor mode. If not, it displays a message and exits.
The script then extracts unique activities from the test data and begins real-time traffic analysis. It enters a loop where each captured traffic is classified, and the activity is displayed on the console along with the elapsed time. This loop continues until the user presses Ctrl+C to stop the script.
Real-time traffic analysis is performed using the *capture_traffic()* function, which captures traffic on the specified interface, extracts features, predicts the activity, and returns it. The activity is added to a list, and the last ten activities are shown on the console, along with their percentage occurrence to provide an idea of the classifier's accuracy.

3.3.2 Auxiliary Functions

3.3.2.1 `list_interfaces`

This function lists all available network interfaces and returns a mapping. The user will use this list to select a network interface.

3.3.2.2 `choose_interface`

This function prompts the user to select an interface from the available list by typing the corresponding number.

3.3.2.3 `check_monitor_mode`

This function checks if the specified interface is in monitor mode. It uses the *iw dev <interface> info* command to retrieve information about the selected interface and checks if "monitor" appears in the output.

3.3.2.4 `print_main_info`

This function calls the *print_main_info()* function from the training and evaluation script to display the main script information and appends a new line with the classifier's accuracy.

3.3.2.5 `print_interface_info`

This function prints information about the chosen interface, including its type (Monitor/Managed). It is called together with the *print_main_info()* function to provide the user comprehensive information about the interface and its type.

3.3.2.6 `handle_wlan_frame`

This function converts the WLAN frame type/subtype from hex to the Wireshark format. It is used when the script prints packets to the console. Although the type/subtype was initially considered to be used for the features, this idea was discarded. The function remains useful for displaying packets in a more readable format.

3.3.2.7 `hex_to_ascii`

This function converts a hex string to an ASCII string. It is used to convert packet SSIDs from hex to ASCII for better readability. Like the type/subtype, the SSID was initially considered to be potentially used for the features, but this idea was also discarded.

3.3.2.8 `save_to_csv`

This function saves data to a .csv file. It is used to store the latest traffic capture in a .csv file, creating a temporary file with the same structure as the test data files. This allows features to be extracted using the same functions from the training and evaluation script.

3.3.2.9 `guess_activity`

This function predicts the activity based on features using the trained model. It uses the encoder to inverse transform the predicted activity and returns it.

3.3.2.10 `capture_traffic`

This is the main function of the script. It captures traffic on the specified interface, classifies the activity, and returns it.
Initially, it waits for the user to press Enter to start capturing traffic. The *pyshark.LiveCapture()* function is then used to capture traffic on the specified interface for the duration specified in the configuration file. For better readability, only packets with the target MAC address as the transmitter, source, receiver, or destination are printed to the console. All packets (also the non-filtered ones) are saved in a .csv file using the *save_to_csv()* function, following the structure of the test data files.
Features are extracted from the .csv file using the *analyze_data()* function from the training and evaluation script, and the temporary .csv file is deleted.
The activity is predicted using the *guess_activity()* function and displayed on the console along with the elapsed time. The function also returns the predicted activity.

3.3.2.11 `extract_unique_activities`

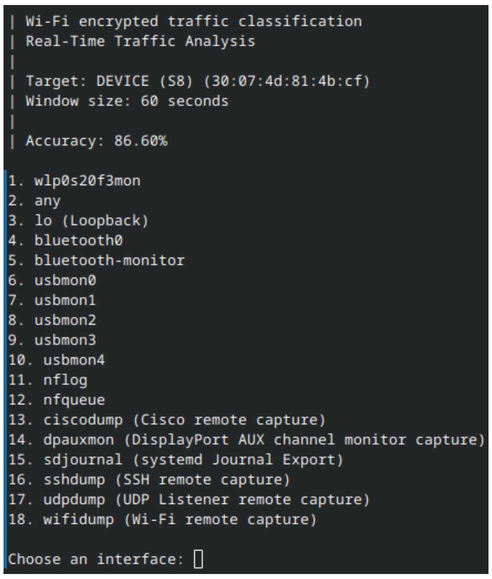
This function extracts unique activities from the .csv files in the specified directory (test_data directory). It is used to gather the unique activities from the test data file names.

3.3.3 Execution Demonstration

Running the script is straightforward. Use Python 3, but ensure you have root privileges:

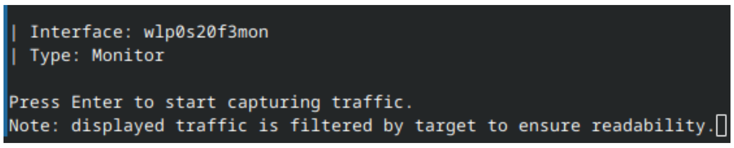
```
sudo python3 real_time_traffic_analysis.py
```

When executed, the script will first run the training and evaluation script to load the classifier and encoder. For more details, refer to the "Execution demonstration" section of the training and evaluation part.
After closing the confusion matrix window opened by the training and evaluation script, the real-time traffic analysis script will prompt you to select a network interface:



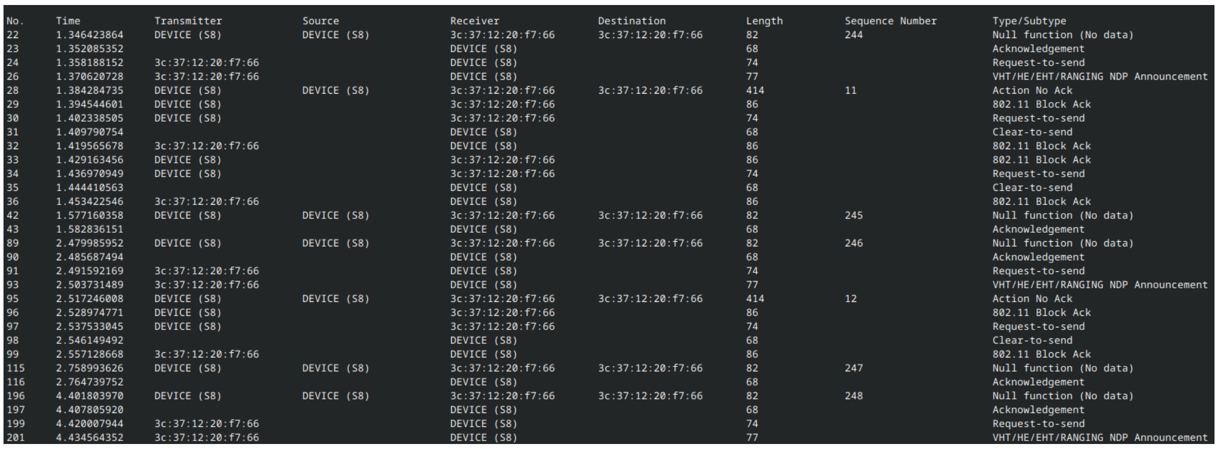
```
| Wi-Fi encrypted traffic classification
| Real-Time Traffic Analysis
|
| Target: DEVICE (S8) (30:07:4d:81:4b:cf)
| Window size: 60 seconds
|
| Accuracy: 86.60%

1. wlp0s20f3mon
2. any
3. lo (Loopback)
4. bluetooth0
5. bluetooth-monitor
6. usbmon0
7. usbmon1
8. usbmon2
9. usbmon3
10. usbmon4
11. nflog
12. nfqueue
13. ciscodump (Cisco remote capture)
14. dpauxmon (DisplayPort AUX channel monitor capture)
15. sdjournal (systemd Journal Export)
16. sshdump (SSH remote capture)
17. udpdump (UDP Listener remote capture)
18. wifidump (Wi-Fi remote capture)

Choose an interface: 
```

You can select an interface from the list by typing the corresponding number. The script will then check if the interface is in monitor mode and display information about the interface type. After choosing the interface, press Enter to start capturing traffic:



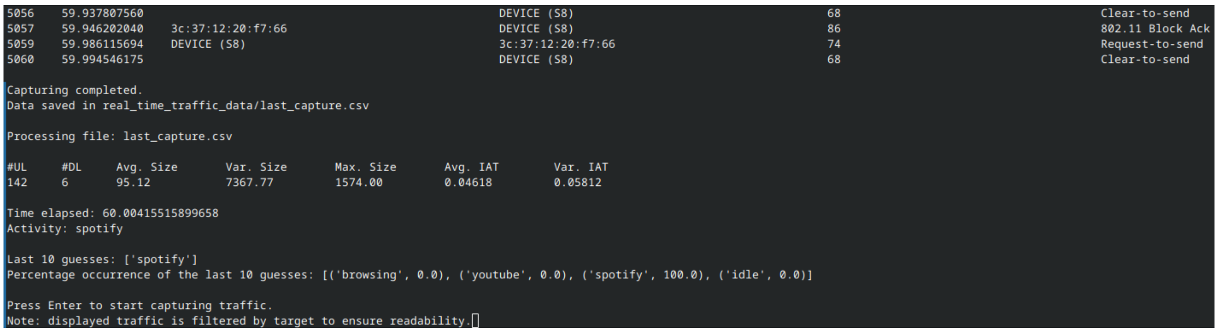
```
| Interface: wlp0s20f3mon
| Type: Monitor

Press Enter to start capturing traffic.
Note: displayed traffic is filtered by target to ensure readability.
```

The script will capture traffic on the selected interface for the duration specified in the configuration file. Packets will be displayed on the console:



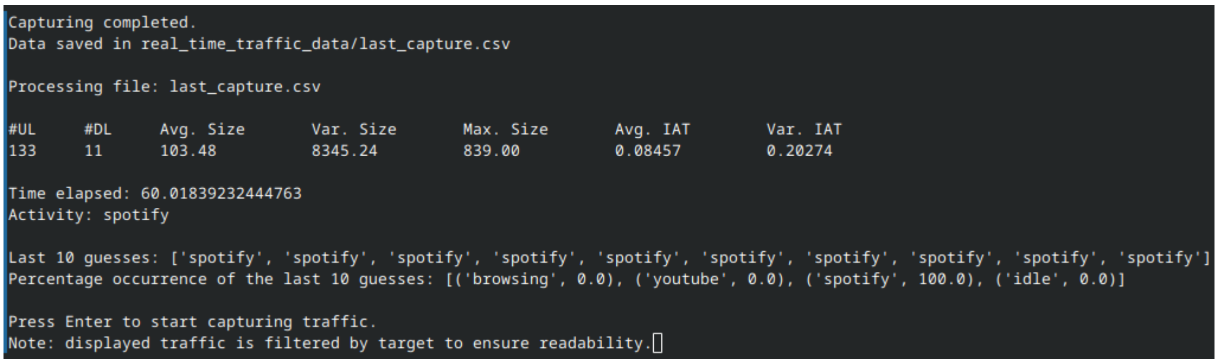
```
No.    Time          Transmitter          Source               Receiver             Destination          Length        Sequence Number      Type/Subtype
22     1.346423864   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    82            244                  Null function (No data)
23     1.352085352                                             DEVICE (S8)                               68                                 Acknowledgement
24     1.358188152   3c:37:12:20:f7:66                         DEVICE (S8)                               74                                 Request-to-send
26     1.370620728   3c:37:12:20:f7:66                         DEVICE (S8)                               77                                 VHT/HE/EHT/RANGING NDP Announcement
28     1.384284735   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    414           11                   Action No Ack
29     1.394544601   DEVICE (S8)                               3c:37:12:20:f7:66                         86                                 802.11 Block Ack
30     1.402338505   DEVICE (S8)                               3c:37:12:20:f7:66                         74                                 Request-to-send
31     1.409790754                                             DEVICE (S8)                               68                                 Clear-to-send
32     1.419565678   3c:37:12:20:f7:66                         DEVICE (S8)                               86                                 802.11 Block Ack
33     1.429163456   DEVICE (S8)                               3c:37:12:20:f7:66                         86                                 802.11 Block Ack
34     1.436970949   DEVICE (S8)                               3c:37:12:20:f7:66                         74                                 Request-to-send
35     1.444410563                                             DEVICE (S8)                               68                                 Clear-to-send
36     1.453422546   3c:37:12:20:f7:66                         DEVICE (S8)                               86                                 802.11 Block Ack
42     1.577160358   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    82            245                  Null function (No data)
43     1.582836151                                             DEVICE (S8)                               68                                 Acknowledgement
89     2.479985952   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    82            246                  Null function (No data)
90     2.485687494                                             DEVICE (S8)                               68                                 Acknowledgement
91     2.491592169   3c:37:12:20:f7:66                         DEVICE (S8)                               74                                 Request-to-send
93     2.503731489   3c:37:12:20:f7:66                         DEVICE (S8)                               77                                 VHT/HE/EHT/RANGING NDP Announcement
95     2.517246008   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    414           12                   Action No Ack
96     2.528974771   DEVICE (S8)                               3c:37:12:20:f7:66                         86                                 802.11 Block Ack
97     2.537533045   DEVICE (S8)                               3c:37:12:20:f7:66                         74                                 Request-to-send
98     2.546149492                                             DEVICE (S8)                               68                                 Clear-to-send
99     2.557128668   3c:37:12:20:f7:66                         DEVICE (S8)                               86                                 802.11 Block Ack
115    2.758993626   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    82            247                  Null function (No data)
116    2.764739752                                             DEVICE (S8)                               68                                 Acknowledgement
196    4.401803970   DEVICE (S8)          DEVICE (S8)          3c:37:12:20:f7:66    3c:37:12:20:f7:66    82            248                  Null function (No data)
197    4.407805920                                             DEVICE (S8)                               68                                 Acknowledgement
199    4.420007944   3c:37:12:20:f7:66                         DEVICE (S8)                               74                                 Request-to-send
201    4.434564352   3c:37:12:20:f7:66                         DEVICE (S8)                               77                                 VHT/HE/EHT/RANGING NDP Announcement
```

The script will classify the captured traffic and display the activity on the console, along with the elapsed time. It will also show the last ten activities and their percentage occurrence:



```
5056    59.937807560                                        DEVICE (S8)                            68                              Clear-to-send
5057    59.946202040    3c:37:12:20:f7:66                   DEVICE (S8)                            86                              802.11 Block Ack
5059    59.986115694    DEVICE (S8)                         3c:37:12:20:f7:66                      74                              Request-to-send
5060    59.994546175                                        DEVICE (S8)                            68                              Clear-to-send

Capturing completed.
Data saved in real_time_traffic_data/last_capture.csv

Processing file: last_capture.csv

#UL     #DL     Avg. Size       Var. Size       Max. Size       Avg. IAT        Var. IAT
142     6       95.12           7367.77         1574.00         0.04618         0.05812

Time elapsed: 60.00415515899658
Activity: spotify

Last 10 guesses: ['spotify']
Percentage occurrence of the last 10 guesses: [('browsing', 0.0), ('youtube', 0.0), ('spotify', 100.0), ('idle', 0.0)]

Press Enter to start capturing traffic.
Note: displayed traffic is filtered by target to ensure readability.
```

Each time you press Enter after classification, the script will capture the traffic again and classify the activity. This process will continue until you press Ctrl+C to stop the script. Here is the result after 10 captures:



```
Capturing completed.
Data saved in real_time_traffic_data/last_capture.csv

Processing file: last_capture.csv

#UL     #DL     Avg. Size       Var. Size       Max. Size       Avg. IAT        Var. IAT
133     11      103.48          8345.24         839.00          0.08457         0.20274

Time elapsed: 60.01839232444763
Activity: spotify

Last 10 guesses: ['spotify', 'spotify', 'spotify', 'spotify', 'spotify', 'spotify', 'spotify', 'spotify', 'spotify', 'spotify']
Percentage occurrence of the last 10 guesses: [('browsing', 0.0), ('youtube', 0.0), ('spotify', 100.0), ('idle', 0.0)]

Press Enter to start capturing traffic.
Note: displayed traffic is filtered by target to ensure readability.
```

As you can see, the script performed pretty well in classifying the activity. In fact, while streaming Spotify, the script consistently identified the activity as "spotify."
Note: If you change activities on the device (e.g., switching from Spotify to YouTube), wait a few minutes before starting the capture again to achieve more accurate results.

4 Conclusions

The project has successfully met the initial requirements and objectives. The classifier performs well in classifying activities, and the real-time traffic analysis generally works as expected. I had fun working on this project and extending it with additional parts that, while not strictly required, I found interesting to implement. This was my first experience with a machine learning project, and I learned a lot not only about machine learning libraries, but also about Wi-Fi traffic and how to capture and analyze it.