Real time Alexa packets profiling analysis

Giuseppe Polese
gpolese@unisa.it
Universitá degli studi di Salerno
Salerno, Italy

Bernardo Breve
Stefano Cirillo
bbreve@unisa.it
scirillo@unisa.it
Universitá degli studi di Salerno
Salerno, Italy

Biagio Boi
b.boi@studenti.unisa.it
Universitá degli studi di Salerno
Salerno, Italy

ABSTRACT

Nowadays, the introduction of home virtual assistents like Alexa Echo or Google Home is becoming a practice, just considering that more than 27% of families own one.

It's obvious that those devices simplify the life by creating a smart house with few money; but what's the impact these devices have on people's privacy? There are a lot of cases in the United States in which the judge asked Amazon to provide the recording done by the Echo in order to find helpful evidences for the case; so, the question is: "Is it possible to prevent the sending of sensible informations to the servers when the wake word is not pronunced?"

In this project we will profile each packet exchanged between Alexa Echo and the Server in order to classify the nature of the packets and consequentially we will use a machine learning model to discover if exists a clear correlation between the characteristics of the packet and the classification done.

KEYWORDS

data analytics, alexa, packets profiling

1 INTRODUCTION

The evolution of smart devices over last years has increased exponentially and the introduction of these devices in the house is progressively growing. The major problem related to these devices is that usually the privacy is not considered, although there are a lot of regulations (just see the GDPR) that describe how the user data have to be stored and who can access to these data. Starting from these two points I decided to understand what happens within the context of smart assistance device such as Alexa Echo. Reading on the Amazon Alexa website it's clear how and when they send data to the Amazon Cloud, but on the other side we have some cases in which the recording done by the Echo have been used during the processes in order to extract some evidences. In order to clarify this aspect and the role of the smart assistance devices I decided to analyze the outgoing traffic from the device and classify it by considering the environmental conditions. The first phase of this analysis consists in understand which kind of packets are sent from the device in order to create a complete dataset of packets sent by the device; during the second phase I tried to understand if it exists a pattern that identify the packets that souldn't be sent to the Cloud by creating a machine learning model able to identify them and consequentially use this model within a script in order to block unauthorized packets.

2 STATE OF ART

During the last years different researches have been done, related to different aspects of the security and privacy within the IoT context. F. Z. Berrehili and A. Belmekki [1] have done a study on the Privacy Preservation in the IoT context that shows all the risks directly related to the architecture; in particular, it underlines the need to always guarantee the anonymization of sensibile data by hiding the correlation between these data and the person who produced these data (that is also one of the principle of GDPR). Despite these studies, there is a huge percentage of devices that does not consider as serious all the aspects related to the privacy, in particular J. Liu and W. Sun [2] show different attacks againit werables devices at different levels of the ISO/OSI stack and considering different aspects of privacy and security like data integrity, authentication, authroization, etc. The study shows a lot of attacks aim at sniff packets during the communication in order to collect sensible informations; this seems to be caused by problems related to poor encryption smechanisms implemented into these smart devices.

3 ALEXA ARCHITECTURE & SECURITY

The Alexa architecture isn't easy to explain, we will resume the keypoints in order to better understand the main functionalities for our purpose.

1. Alexa is always in listening waiting for the wake word to be pronunced to start the recording of voice;
2. From the wake word, till the end of commands, Alexa will record the speech and partially sends it to Alexa Voice Service, that can be considered as the brain of Alexa;
3. Alexa Voice Service will process the audio using Natural Language Processing and Natural Language Understanding in order to retrieve a response for the given request.
   (a) Natural Language Processing (NLP) improve the Word Segmentation that separate a chunk of continuous text into separate words.
   (b) Natural Language Understanding (NLU) is a subtopic of NLP and uses the AI to map text to the meaning[3] in order to understand the speech and the request.
4. Depending on the sent command, the Voice Service will take an action (turn on the light) or send the information back to the device and Alexa may speech.

All the communications between Echo and Server are secured by using an SSL/TLS encryption schema. More throughly, Echo looks for an available Amazon Server every 2 minutes and then establishes a connection (by following the SSL Handshake). Once the communication has been initialized, Echo starts to send different encrypted packets of different size; clearly, these pakcets may contains both authorized and unauthorized data. It may be important to underline that all recordings are available on Amazon platform and can be reproduced and deleted whenever the user wants only by accessing to the platform. This accessibility to data guarantees more transparency to the user that can directly handle all recordings. The point is that here we found only recordings done by Echo in an authorized way (when the wake word it's been pronounced), but this doesn't imply that some other recordings can exist and may be stored in hidden way.

4 PACKETS ANALYSIS

In order to analyze packets we need to create some kind of proxy between Echo and Router (that will send packets to the server); for this reason Echo it's connected directly to the computer using the Windows mobile hotspot.

4.1 Python script

The step described above is necessary since we need to execute a python script on the same machine that recieves packets from Echo. In particular, the script will intercept every packet sent by Echo using PyShark library, that is a library developed in python based on the popular software Wireshark, able to intercept every kind of packet and to retrieve useful information about them. The first step before using the library is to configure it by setting up the preferred virtual network card (in our case will be that on which is run the hotspot) and the device we want to sniff the packets. Once these two parameters have been declared, the sniffing can start: we will collect only those packets sent by Echo, by discarding the others (those recieved from the Server); this choice it's been taken since we don't want to block or discover anything about the recieved packets, we want only to block those packets that shouldn't be sent. A simple python function is called every time a new packet has been sent, it processes packets by checking at: protocol, flags, content type and length. During the processing phase, the packets are classified and inserted into the relative dataset (one for each context in which the script has been executed). The dataset created by the script needs a manual check in order to detect errors.

4.2 Classification

It's possible to classify packets by looking at flags, packet size and protocol used. There are different types of packest that will later be included in the dataset as classes:

1. **handshake**: At the beginning of each new SSL/TLS communication between Echo and Server there are different packets exchanged in order to establish a secure communication. These packets can be easily identified by the protocol used for the communication (TLS) and by checking the flags related to the content of the message, in particular, the possible flags are:
   (a) Change Cipher (20).
   (b) Server Hello - Show Certificate - Encrypted Message (21).
   (c) Alert (22).
2. **syn**: Packets with fixed length are sent from the device in a fixed interval, it seems that them are used to synchronize Echo with the Server.
3. **ack**: These are the classical packets used to confirm that the received packets are valid and successfully received.
4. **retransmit**: These packets are used to retransmit data that didn't received an ack, usually this happens when Echo tries to communicate with other Amazon devices (Fire Stick for ex.) but doesn't receive response.
5. **app_data**: These packets are relative to the normal communication of Echo and can be easily recognized since the communication happens over TLS/SSL protocol and by checking the flag of the considered packet that is equal to 23. In particular, we will include in this class all the communication packets that include also recording of voice occurred during the conversation between Echo and the final costumer.
6. **not_relevant**: We will include in this category all packets that don't match any of the other categories, and are not relevant for our study.

It may be import to underline that the packets classified as app_data may include both authorized and not authorized packets. The unauthorized packets include all those packets sent in a context in which no application date should be sent to the Server by considering the environmental variables (no wake word pronounced) and hardware aspects (microphone muted by pressing the relative button); for this reason it is important to introduce some classes related to these aspects. The aim is to distinguish three kind of packets:

- **not_justified**: packets sent to the Server in a context in which Echo is unable to send these data by considering hardware aspects.
- **justified**: packets sent to the Server without any interaction between Echo and user, but Echo is able to listen.
- **expected**: packets sent to the Server since an interaction between Echo and user happened.

Clearly, the only allowed packets are those classified as "Expected".

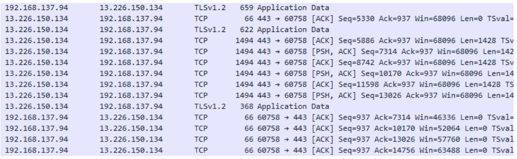
4.3 Consideration on packet analysis

It's important to do some consideration over the analysis / collection phase just described, in fact there are a lot of behavioural pattern that the Echo follow to perfrom some actions:

- As introduced in the previous section there are some kinds of packets that always follow the same schema in a given interval. These packets may be sent to guarantee the synchronization with the Amazon Alexa application and with Amazon servers. There are two possible synchronization packets:
  1. A packet of 100 byte is sent every 30 seconds (30002 milliseconds)
  2. A packet of 99 byte is sent ebery 90 seconds (90820 milliseconds)

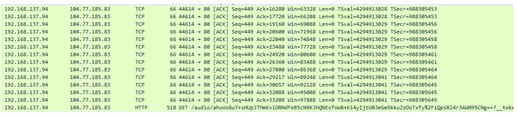
  The communication always happens using SSL/TLS, so it's impossible to decrypt the content of these message and confirm that is securely a synchronization packet.
- Music stream pattern is dependent on music service used; a stream with Amazon Music always uses a secure connection by using TLS.

  Instead, a stream with Spotify makes an HTTP request to Spotify Server that can be captured and replicated



```
192.168.137.94    13.226.150.134    TLSv1.2   659 Application Data
13.226.150.134    192.168.137.94    TCP        66 443 → 60758 [ACK] Seq=5330 Ack=937 Win=68096 Len=0 TSval=
13.226.150.134    192.168.137.94    TLSv1.2   622 Application Data
13.226.150.134    192.168.137.94    TCP      1494 443 → 60758 [ACK] Seq=5886 Ack=937 Win=68096 Len=1428 TSv
13.226.150.134    192.168.137.94    TCP      1494 443 → 60758 [PSH, ACK] Seq=7314 Ack=937 Win=68096 Len=142
13.226.150.134    192.168.137.94    TCP      1494 443 → 60758 [ACK] Seq=8742 Ack=937 Win=68096 Len=1428 TSv
13.226.150.134    192.168.137.94    TCP      1494 443 → 60758 [PSH, ACK] Seq=10170 Ack=937 Win=68096 Len=14
13.226.150.134    192.168.137.94    TCP      1494 443 → 60758 [ACK] Seq=11598 Ack=937 Win=68096 Len=1428 TS
13.226.150.134    192.168.137.94    TCP      1494 443 → 60758 [PSH, ACK] Seq=13026 Ack=937 Win=68096 Len=14
13.226.150.134    192.168.137.94    TLSv1.2   368 Application Data
192.168.137.94    13.226.150.134    TCP        66 60758 → 443 [ACK] Seq=937 Ack=7314 Win=46336 Len=0 TSval=
192.168.137.94    13.226.150.134    TCP        66 60758 → 443 [ACK] Seq=937 Ack=10170 Win=52064 Len=0 TSval
192.168.137.94    13.226.150.134    TCP        66 60758 → 443 [ACK] Seq=937 Ack=13026 Win=57760 Len=0 TSval
192.168.137.94    13.226.150.134    TCP        66 60758 → 443 [ACK] Seq=937 Ack=14756 Win=63488 Len=0 TSval
```

Figure 1: Captured packets during Amazon Music streaming (Wireshark)



```
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=16280 Win=63328 Len=0 TSval=4294913028 TSecr=988305453
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=17720 Win=66208 Len=0 TSval=4294913028 TSecr=988305453
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=19160 Win=69088 Len=0 TSval=4294913029 TSecr=988305456
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=20600 Win=71968 Len=0 TSval=4294913029 TSecr=988305456
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=22040 Win=74848 Len=0 TSval=4294913029 TSecr=988305458
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=23480 Win=77728 Len=0 TSval=4294913029 TSecr=988305458
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=24920 Win=80608 Len=0 TSval=4294913029 TSecr=988305461
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=26360 Win=83488 Len=0 TSval=4294913029 TSecr=988305461
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=27800 Win=86368 Len=0 TSval=4294913029 TSecr=988305464
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=29217 Win=89248 Len=0 TSval=4294913041 TSecr=988305464
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=30657 Win=92128 Len=0 TSval=4294913041 TSecr=988305645
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=32088 Win=95008 Len=0 TSval=4294913041 TSecr=988305645
192.168.137.94    104.77.185.83    TCP     66 44614 → 80 [ACK] Seq=449 Ack=33208 Win=97888 Len=0 TSval=4294913041 TSecr=988305645
192.168.137.94    104.77.185.83    HTTP   518 GET /audio/ahuVo8u7+sHUpJTMmKviORNdFeB5cN9KJhQNEsfoW8+Ki4yIjtU0JmSm5kkuZoOoTxfy%2FiQpsRl4r3AWN95CNg==?__toke
```

Figure 2: Captured packets during Spotify streaming (Wireshark)

- Echo changes server every 2 minutes, for this reason there are a lot of handshake packets.

5 DATASET

In order to create a valid and well-distributed dataset we analyzed the behavior of Echo in different context, in particular by considering different cases:

(1) **Microphone mute**: when the hardware button is pressed and the device should be unable to send application data to the Server.

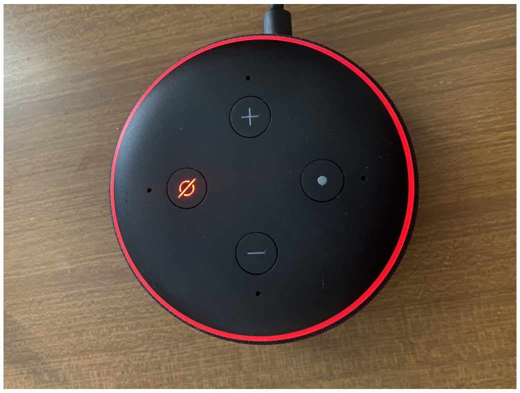


Figure 3: Echo when the disable microphone button has been pressed.

(2) **No wake word pronounced**: when the device is able to listen the voice but shouldn't send to the Server any packets of application data.
(3) **Normal stream of data**: when a communication happens. It may include different questions or requests, for example:
  (a) How is the weather today?
  (b) When does Liverpool play?
  (c) Add milk to the shopping list.
  (d) How much is 20 + 20?
  All these questions or requests expect a response from the Server, so a huge amount of ack will be sent from the device.
(4) **Streaming**: when the user requests for a song and Echo starts to stream it; also in this case a lot of ack packets are sent from the device as response to the fragment of the item to stream (a song for example).

5.1 Features

As introduced in the previous section it's important to collect features related to both hardware and software aspects; let's see in detail which are the considered features with the related meaning:

(1) **date**: the date in which the packet has been collected, in the format YYYY-MM-DD hh:mm:ss:ms;
(2) **length**: the length of the packet, it includes just the payload (in case of ack packest it's equal to zero);
(3) **dstip**: the destination ip, collected to analyze the owner of the server to which the packet is directed;
(4) **dstport**: the destination port;
(5) **highest_layer**: the protocol used, in order to parse the protocol into an integer we will use the following mapping:
  - 0 - SSL
  - 1 - TCP
  - 2 - DATA
  - 3 - HTTP

  Notice that all the packets that use other protocol are discarded since they have no meaning for our purpose;
(6) **delta**: the time occured from the previous packet of the same stream;
(7) **ack_flag**: the acknowledge flag; it is equal to 1 if the packet contains an ack;
(8) **microphone**: the status of the microphone, it is equal to 1 if the microphone is active, 0 otherwise;
(9) **content_type**: the type of content sent, it is valorized only if the packet is sent over SSL;
(10) **synchronized**: status of sychronization of the device, it is equal to 1 if Echo has been previously associated to an account, 0 otherwise.

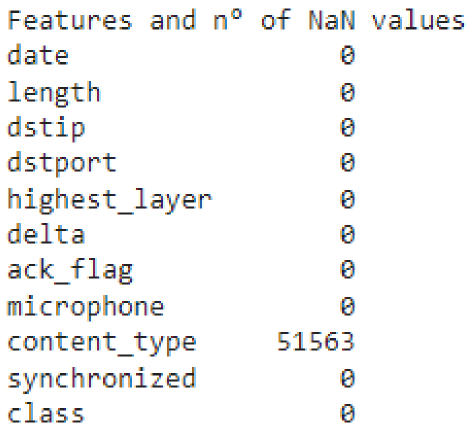
Since we want to map every kind of packet in a class, 8 classes have been declared: handshake, syn, ack, retransmit, not_relevant, justified, not_justified and expected.

5.2 Dataset merging

Since we have collected packets into four different contexts is necessary to merge together all these dataset before to apply any kind of manipulation on data. In particular during the collection phase we have collected 5436 packets when the microphone has been disabled, 39793 packets when the music was streaming (indipendently by the service used), 9374 packets when no wake word has been pronounced and 8027 packets during a normal conversation with Echo. Clearly, a first overview on these numbers may start thinking that it's totally unbalanced; but we should remember that during the streaming of contents there is an huge amount of ack packets that can be potentially discarded; we will see this operation in the following subsection.

5.3 Avoid NaN values

During a first overview on collected packets it's been noticed that there are some packets in which the content_type feature is set to NaN. This problem has been caused by the collecting phase; in fact, as already introduced in the subsection 5.1, this feature is valorized only when we capture an SSL packet, otherwise it is set to NaN. In order to fix this problem, all the existing NaN values for the content_type feature has been set to 0.0.



```
Features and n° of NaN values
date                0
length              0
dstip               0
dstport             0
highest_layer       0
delta               0
ack_flag            0
microphone          0
content_type    51563
synchronized        0
class               0
```

Figure 4: NaN values related to the content_type feature.

5.4 Not relevant features

The dataset contains also instance-specific information, like date and destination ip of the pakcets. Since these values cannot be used during the training phase (otherwise the model will use these values to predict the class of the packets) we need to completely discard these features by deleting the column from the dataset.

5.5 Dataset analysis

After removed the NaN values and the not relevant features it is important to analyze the collected packets in order to understand if they can be already used or if exist some other correction that must be applied before the training. The analysis is divided into target and features analysis.

5.5.1 Target class analysis. Let's start the analysis of the target class by understanding how many packets have been collected for each class.

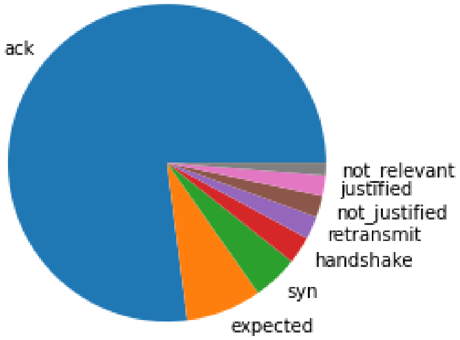


Figure 5: Distribution of target classes.

It's clear the presence of too many ack packets compared to the other classes; as said before this is caused by the streaming context; in fact when a song is streaming Echo will send a lot of ack corresponding to each fraction of the song that has been recieved from Echo. The total amount of ack packets is 48252, compared to: 4813 for expected; 2888 for syn; 1751 for handshake; 1457 for retransmit; 1361 for not_justified; 1316 for justified and 792 for not_relevant. Translating into percentage, the number of ack packets corresponds to 77% over the total. In order to make more balanced the overall dataset we will consider just the 5% of these packets, that is equal to 2413. The deletion of the row is random.

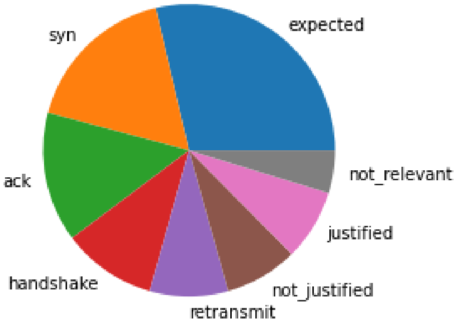


Figure 6: Distribution of target classes after 95% ack packets delete.

5.5.2 Features analysis. After guaranteed a good distribution for the target classes, we start our analysis on the features. Let's see the distribution of length of packets in correlation with the target classes.

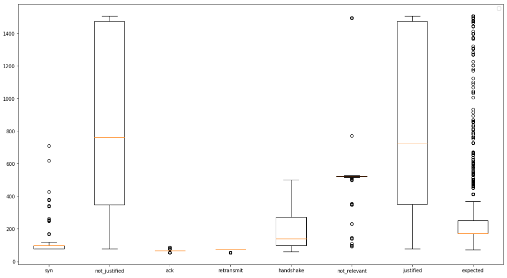


Figure 7: Distribution of length in correlation with target classes.

It's clear that synchronization and acknowledge packets have a short length compared to the others; in fact these packets have small data within them, a simple text or just a flag. Instead, application data packets such as not justified, justified and expected packets have a variable longer length; in fact these packets are usually used to send voice recording for example. A more detailed view of the length of these packets can be seen by plotting the length grouped by the category.

Another important consideration is that there is a small difference between the expected and the justified / not justified packets; the first ones usually have a smaller length, while the others have similar behaviour between each other but the average of length is much higher compared to the expected ones. Anyway we are not worried about the difficulty for the classifier to distinguish the justified ones from the not justified ones; in fact we know that any justified packet exists with the microphone off (by the assumption made during the collection phase).

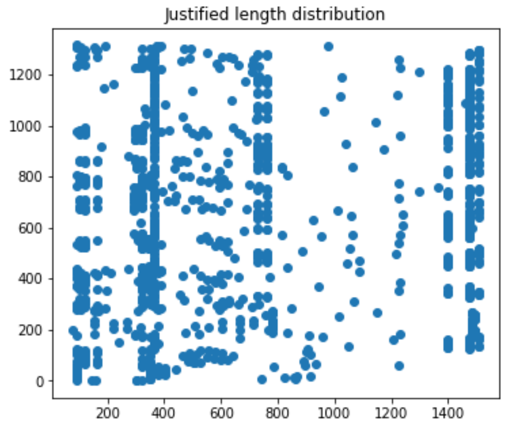


Figure 8: Distribution of length of justified packets.

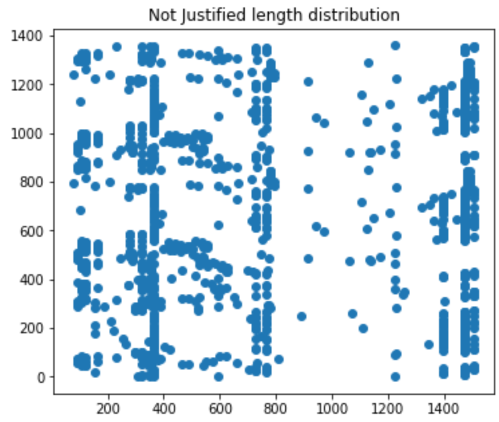


Figure 9: Distribution of length of not justified packets.

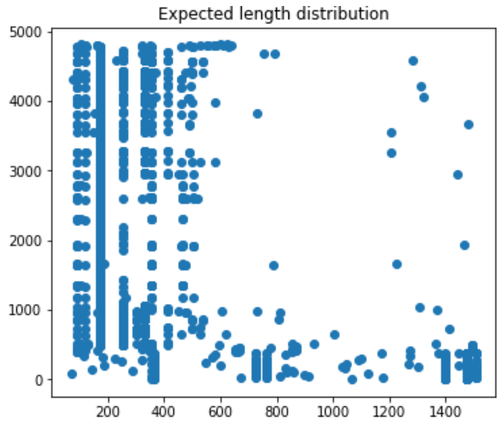
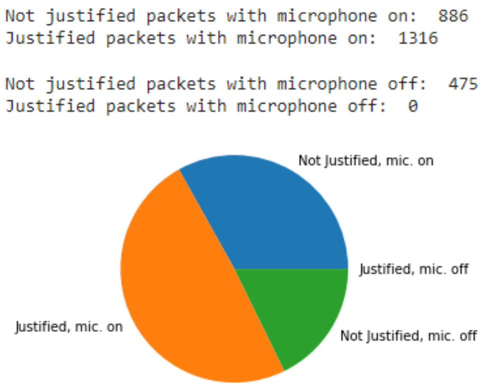


Figure 10: Distribution of length of expected packets.

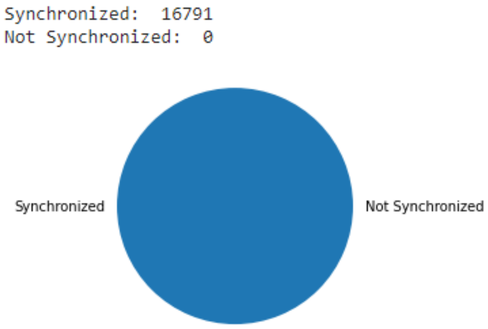
Notice that this behaviour is not exclusive each other, in fact some not justified packets may exist when the microphone is on and nothing is happening in the external environment.



```
Not justified packets with microphone on:  886
Justified packets with microphone on:  1316

Not justified packets with microphone off:  475
Justified packets with microphone off:  0
```

Figure 11: Distribution of justified and not justified packets in correlation with microphone on/off.

Till now, we did not talk about the synchronized feature; this feature figures out the state of synchronization of the device (if it is associated with an Amazon account or not). The problem here is that Echo cannot work if it is not synchronized with any account, in fact if we plot a pie chart of the distribution of this feature we can easily see that all the captured packets have been captured when the device was synchronized.

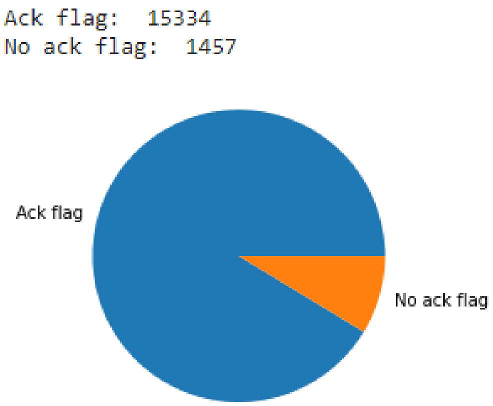


```
Synchronized:  16791
Not Synchronized:  0
```

Figure 12: Distribution of synchronized packets.

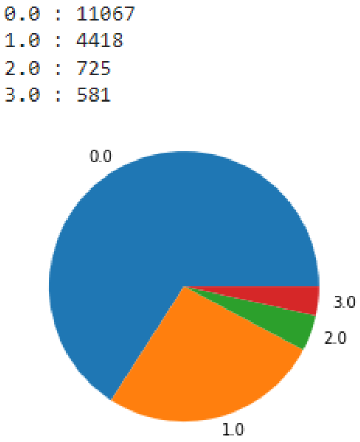
For this reason we have to delete also this feature (it isn't relevant for our purpose). One of the last analysis regards the distribution of ack flag; in fact this flag seems quite always set to 1 (usually a communication packet may contains also an acknowledge of the previous recieved packet) but we decide to mantain this feature since there is a 10% of packets in which this flag is set to 0, so it is useful for the training.

Last analysis regards the distribution of highest_layer feature. Here we have a lot of SSL packets because this is the major protocol used for the communication between Echo and Server; while TCP protocol is used for the acknowledge packets and the others (data and http) are used only when a song is requested to the spotify server or for others synchronization features.



```
Ack flag:  15334
No ack flag:  1457
```

Figure 13: Distribution of ack packets.



```
0.0 : 11067
1.0 : 4418
2.0 : 725
3.0 : 581
```

Figure 14: Distribution of highest layer.

6 MACHINE LEARNING

Once the analysis of the dataset is ended we can start the study of the machine learning approaches in order to retrieve which is more suitable for our purpose.

6.1 Data balancing

Before normalizing the data and apply other machine learning techniques we need to balance the dataset; as we have seen in the previous section there is a small unbalancing between expected and synchronization packets compared to the others; for this reason we will apply a SMOTE (Synthetic Minority Oversampling Technique) in order to oversample the packets of the minority classes. SMOTE will repeat the oversampling procedure till the minority classes will reach the proportion of the majority classes. After SMOTE applied, we have a dataset composed by 4813 packets for each class.

6.2 Dataset splitting and standardization

Once we have balanced the dataset, we need to split it into testing and training dataset. The splitting is done by using the *train_test_split* technique, which is implemented into the sklearn library and split the dataset using a randomic approach by starting from test_size and random_state parameter. For our purpose we set the test_size to 0.3, which means the 30% over the total and random_state equal to 6. After that splitting is completed, we need to standardize the existing values in order to normally distribute them, in particular a standard scaler has been applied.

6.3 Features evaluation

As we will see below, the feature evaluation has been done by applying a random forest classifier (which better fits for our purpose) and then by analyzing the *feature_importances* attribute of the classifier. The report shows that the length it's the most useful feature of the overall dataset as we did previously see during the analysis phase of the project; in fact it depends on the kind of packet, meaning that there is a clear correlation between each other. The second important feature is the content type, that is useful to distinguish the application data from the hanshake packets. A further important result highlited from this graph is the low importance of ack flag, this because there are a lot of packets classified as application data that have the ack flag equal to 1 (due to architecture of TLS communication which implies the existence of an ack packet together with the application data) that makes difficult for the classifier to understand which are the simple ack packets and which are application data ones.

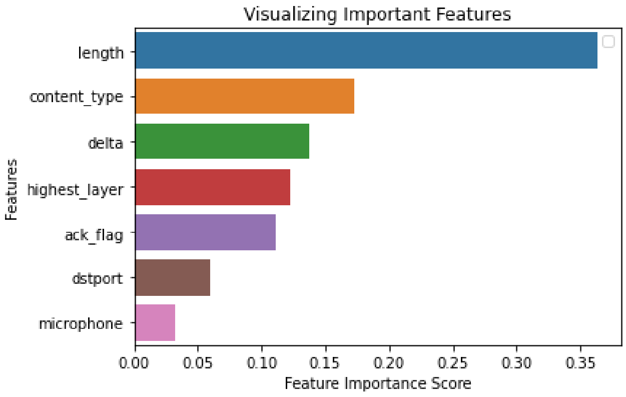


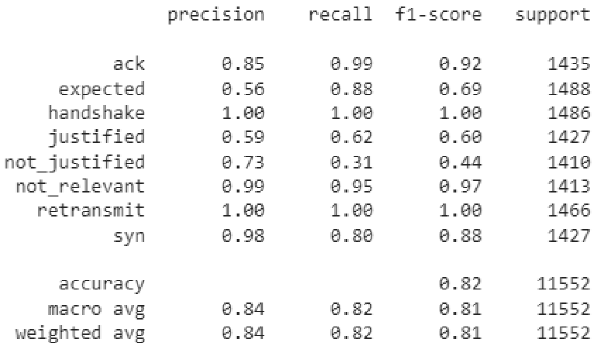
Figure 15: Report of features evaluation.

6.4 Multiclass classification

Since we need to classify instances into one of eight classes we need a multiclass classification, let's see in a more detailed way the possible strategies.

6.4.1 Trasformation to binary. One possible strategy is to transform the problem of multiclass classification to multiple binary classification problems; this transformation can be done in two ways.

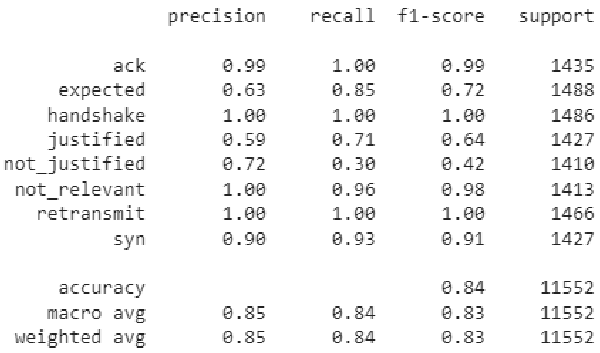
One-vs-rest: we will create a single classifier for each class and the output will be a real-valued confidence score for its decision. The *sklearn.multiclass* library will help us with the creation of this classifier; in particular the class OneVsRestClassifier will create a classifier for each class. The constructor needs an estimator that is an object implementing fit and decision functions; for our purpose we will use a Linear Support Vector Classification (SVC) with random_state parameter equal to 6. It's clear how the classifier easily obtains an accurate precision for acknowledge, handshake, not_relevant, retransmit and synchronization packets due to their easily identification by using flags, content type and highest layer features. While for the application data packets (justified, not justified and expected) the classifier is not able to easily identify the class due to the similar characteristics of these packets. Other problems related to this strategy comes from the definition of this kinf of classifier; since we create a single classifier for each class and the prediction happens by using a binary classification; there is an unbalanced distribution because the set of positives in much smaller than the positive ones.



```
               precision    recall  f1-score   support

          ack       0.85      0.99      0.92      1435
     expected       0.56      0.88      0.69      1488
    handshake       1.00      1.00      1.00      1486
    justified       0.59      0.62      0.60      1427
not_justified       0.73      0.31      0.44      1410
 not_relevant       0.99      0.95      0.97      1413
   retransmit       1.00      1.00      1.00      1466
          syn       0.98      0.80      0.88      1427

     accuracy                           0.82     11552
    macro avg       0.84      0.82      0.81     11552
 weighted avg       0.84      0.82      0.81     11552
```

Figure 16: Report of one vs rest classifier.

One-vs-one: the other possible strategy is one-vs-one; the implementation is quite similar to one-vs-rest approach but there is a difference on the reduction of the overall training set; it creates $k(k-1)/2$ classifiers, where k is the number of existing classes, always by using an esimator object, and each of them recieves the samples of a pair of classes of the original training set and must learn to distinghuish these two calsses. At prediction time, all the $k(k-1)/2$ classifiers will recieve an unseen sample and the class that recieves the highest number of votes will be predicted from the overall classifier. Here there is a better precision for the expected class but it is not enough to say that this one is the best classifier.



```
               precision    recall  f1-score   support

          ack       0.99      1.00      0.99      1435
     expected       0.63      0.85      0.72      1488
    handshake       1.00      1.00      1.00      1486
    justified       0.59      0.71      0.64      1427
not_justified       0.72      0.30      0.42      1410
 not_relevant       1.00      0.96      0.98      1413
   retransmit       1.00      1.00      1.00      1466
          syn       0.90      0.93      0.91      1427

     accuracy                           0.84     11552
    macro avg       0.85      0.84      0.83     11552
 weighted avg       0.85      0.84      0.83     11552
```

Figure 17: Report of one vs one classifier.

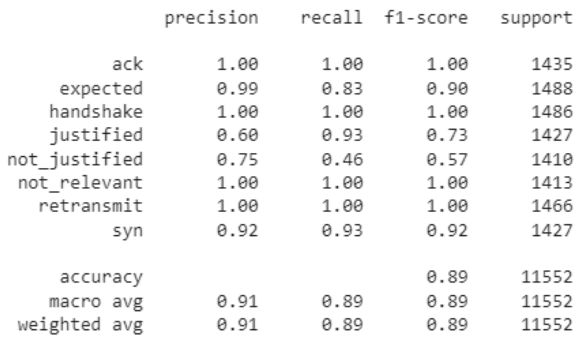
6.4.2 *Extension from binary.*

The other possible strategy to solve the problem of multiclass classification is using existing binary classifiers to solve multiclass problems.

Neural networks: the main differences between neural networks for binary and multiclass classification are:

- **Output layer**: instead of using an output layer of just one neuron, we will use one neuron for each class. Another modification to apply on the output layer is the activation function, using the *softmax* function (instead of *sigmoid*) each neuron yields a probability for corresponding class.
- **Loss function**: instead of using *binary_crossentropy*, we will use the *categorical_crossentropy* that penalizes error in the probabilities predicted by a multiclass classifier (while the *binary* does for binary classifier).

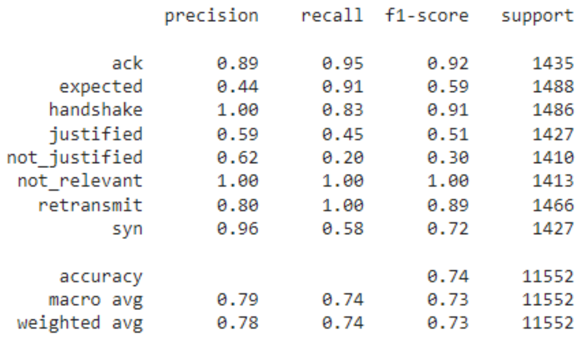
Once underlined the main differences we need to adapt the dataset to be compatible with the neural network; in particular we need to encode the label (classes) since neural network doesn't work with string but just with integer. First, using the class *LabelEncoder* we map each label into an integer; then we create the relative matrix using the *np_utils.to_categorical* method which, for each row, creates a list of 0, except for the position relative to the integer of the relative label. The method returns a matrix composed of n row and m columns; where n is the number of samples and m is the number of classes. The sequential model used for the neural network is composed by four layers (one for input and one for output included). After different observations seems that 250 is the best number of epochs for our classifier. The result highlights that till now the Neural Network classifier is better than the others. In particular, it shows a big difference between the expected and the not justified and justified packets; that is what we need for our purpose.



```
               precision    recall  f1-score   support

          ack       1.00      1.00      1.00      1435
     expected       0.99      0.83      0.90      1488
    handshake       1.00      1.00      1.00      1486
    justified       0.60      0.93      0.73      1427
not_justified       0.75      0.46      0.57      1410
 not_relevant       1.00      1.00      1.00      1413
   retransmit       1.00      1.00      1.00      1466
          syn       0.92      0.93      0.92      1427

     accuracy                           0.89     11552
    macro avg       0.91      0.89      0.89     11552
 weighted avg       0.91      0.89      0.89     11552
```

Figure 18: Report of Neural Network classifier.

Naive Bayes: this approach is based on the probability, there are different applications of Naive Bayes algorithm. Each of these application can be run without any parameter, but if a prior probability is defined, it will fit better. Kernel Density Estimation is a non-parametric way to estimate the probability density function of a random variable. We will use a Gaussian Naive Bayes classifier for our purpose. It's clear how the results are worst than the Neural Network; this is also caused by the fact that Gaussian cannot work with negative values, which exist in our normalized dataset; the

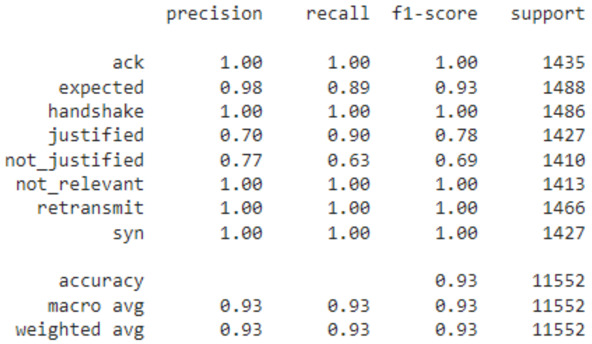


```
               precision    recall  f1-score   support

          ack       0.89      0.95      0.92      1435
     expected       0.44      0.91      0.59      1488
    handshake       1.00      0.83      0.91      1486
    justified       0.59      0.45      0.51      1427
not_justified       0.62      0.20      0.30      1410
 not_relevant       1.00      1.00      1.00      1413
   retransmit       0.80      1.00      0.89      1466
          syn       0.96      0.58      0.72      1427

     accuracy                           0.74     11552
    macro avg       0.79      0.74      0.73     11552
 weighted avg       0.78      0.74      0.73     11552
```

Figure 19: Report of Naive Bayes classifier.

only accepted dataset is the original one, on which there is any kind of normalization technique.

Decision Trees: the goal here is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. One application of Decision Tree is the Random Forest classifier, which fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. We need to define the number of trees in the forest, by default it is set to 100 but after some experiments it seems that 500 perform better. In this case we can use the scaled dataset but in order to show the results a label parsing is needed (from string to integer). Results seem to be better than the Neural Network, this behaviour is justified from the definition of the random forest; it tries to reduce the overfitting, that is really useful for our data (since we have variable length for the same kind of packets). The percentage of accuracy of the three application data packets it's increased and we can easily say that this is the best classifier for our purpose.

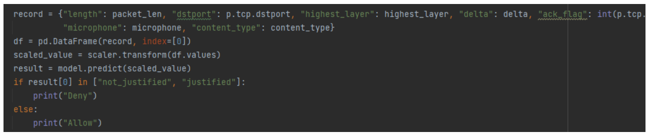


```
               precision    recall  f1-score   support

          ack       1.00      1.00      1.00      1435
     expected       0.98      0.89      0.93      1488
    handshake       1.00      1.00      1.00      1486
    justified       0.70      0.90      0.78      1427
not_justified       0.77      0.63      0.69      1410
 not_relevant       1.00      1.00      1.00      1413
   retransmit       1.00      1.00      1.00      1466
          syn       1.00      1.00      1.00      1427

     accuracy                           0.93     11552
    macro avg       0.93      0.93      0.93     11552
 weighted avg       0.93      0.93      0.93     11552
```

Figure 20: Decision Tree classifier.

7 REAL TIME CLASSIFIER

Once we found a way to classify the packets by using a random forest classifier, we can use it together with a python script in order to classify the packets in real time. The script is similar to that one defined during the collection phase; there is a sniffer that captures all the packets sent from Echo and using the classifier decides the category of the packet. By applying a simple check on the result of the prediction we can block or allow the packet (if it's equal to not justified or justified we will block it). Script performance are very high, we can guarantee a real time classifier able to block the packets without any delay.



```
record = {"length": packet_len, "dstport": p.tcp.dstport, "highest_layer": highest_layer, "delta": delta, "ack_flag": int(p.tcp.
          "microphone": microphone, "content_type": content_type}
df = pd.DataFrame(record, index=[0])
scaled_value = scaler.transform(df.values)
result = model.predict(scaled_value)
if result[0] in ["not_justified", "justified"]:
    print("Deny")
else:
    print("Allow")
```

Figure 21: Piece of real time analyzer.

8 CONCLUSIONS

The aim of the project has been achieved; results confirm the possibility to distinguish between different kinds of packets. In particular, as introduced during the collection phase, both the justified and not justified packets shouldn't be sent from Echo (the first ones because there is no motivation for Echo to sent them even if it can record voice due to the microphone status; while the second ones shouldn't be sent because the device is not listening the voice); a more accurate analysis on the nature of these packets can be done, it may be possible that the packets sent when the device has microphone off are allowed since they might be simple synchronization packets without a clear pattern. Anyway, this study shows that there is a clear behaviour of the expected packets compared to the others; these packets have a lower payload length (it seems that Echo sends an huge amount of short length packets instead of small amount of long length when the communication is "authorized"). The final Random Forest classifier, together with a script that takes into account the consideration made during the collection phase, can be applied on a proxy to block all the packets classified as justified and not justified and see if Echo continues to work well or there are some configuration errors.

REFERENCES

[1] Fatima Zahra Berrehili and Abdelhamid Belmekki. *Privacy Preservation in the Internet of Things*, volume 397. 11 2017.

[2] Jiajia Liu and Wen Sun. Smart attacks against intelligent wearables in people-centric internet of things. *IEEE Communications Magazine*, 54(12):44–49, 2016.

[3] Paul Semaan. Natural language generation: An overview. *Journal of Computer Science & Research (JCSCR)*, 1(3):50–57, 2012.