# POLITECNICO DI TORINO

**MASTER's Degree in COMPUTER ENGINEERING**


**MASTER's Degree Thesis**

# REAL-TIME IDENTIFICATION OF VoIP APPLICATIONS TRAFFIC

**Supervisors**

**Prof. GUIDO MARCHETTO**

**Candidate**

**ANDREA PAVANO**

**ACADEMIC YEAR 2019-2020**

# Summary

This work addresses the problem of real-time recognition of two VoIP applications (Skype and Zoom) via the analysis of the generated traffic. This type of problem, in fact, lately attracted the attentions of the experts in field since its resolution could prove to be quite useful for a series of purposes, like automatic blocking or filtering of traffic flows or labeling of flows for QoS enforcement.
The thesis is structured as follows: after the first chapter that will serve as an introduction, the second chapter will talk about related work, which solutions have already been tested and which levels of accuracy have been reached, and what is the current state of the art; the third chapter will go into more details about the characteristics of the traffic generated by the two applications under analysis; lastly, the fourth chapter will talk about the classifier, how does it work and what are the modules that realize it, while the fifth chapter shows testing results and accuracy values. A conclusive chapter about open issues, possible enhancements and final considerations follows.
An appendix was included to briefly explain machine learning fundamentals and the two models used in this work (Decision Tree and Naïve Bayes).

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In the last few years we witnessed a growing interest in the VoIP technologies, and subsequently the birth of new applications and an ever increasing usage of the already existing ones that exploit this type of technologies.
Many lead the birth of VoIP back to February of 1995 when the company Vocaltec, Inc. released InternetPhone that allowed two users to speak with each other via their computers [1]. It was also in this year that Intel, Microsoft and Radvision initiated standardization activities for VoIP communications system [2].
Since then, the number of VoIP software users kept increasing, to the point that in 2008 VoIP international voice traffic represented 25% of the total of international voice traffic [3].
The reasons behind the success of VoIP were numerous:

- a smaller cost-per-call, even more noticeable for what concerns international calls
- a smaller cost for the infrastructure, since the same IP network already used for other purposes can be reused
- the possibility of adding more functions, even without any change in the hardware
- all the benefits behind the use of the packet switched technologies (e.g. no need to reserve resources in advance)

Because of this, along with many private users even most multinational companies adopted VoIP, mainly in an attempt to bring down the communications costs.
Clearly, along with all the pros of packet switched technologies, VoIP borrows even the cons: in fact, the main problems are that are no guarantees about the in-order arrival of packets, about the delivery of all the packets, about the delay and the variability of it and more in general about the offered QoS.

These problems can be partially tackled by the adoption of the MPLS protocol. But even considering these problems, users are willing to accept an at least decent service for free, also because of the convenience of being able to have access to a certain number of different applications from the same terminal.

Amongst all the available VoIP applications Skype [4] is beyond doubt one of the most notable example of this new phenomenon.
The software, released in 2003 by the same developers of KaZaa (a populare file-sharing client), makes use of a proprietary protocol and offers users many services like instant messagging, file transfers, and of course voice and video calls, all based on a peer-to-peer network. Besides, it is also possible to make calls with traditional phones and send sms to mobile phones.
The importance of Skype is testified by the fact that about 663 millions of users were registered as of 2011.

Another notable example that saw a significant increase in the number of users in the last few years (especially in the beginning of 2020 because of the adoption of quarantine measures due to the COVID-19 pandemic) is Zoom Video Communications [5].
The company, founded in 2011 by Eric Yuan, a former Cisco Webex engineer, launched the software in 2013.
The main services offered by Zoom are online chat and videotelephony, both provided through a cloud-based peer-to-peer platform. The strong points of the application, the main ones being user friendliness and lower costs than other competitors, resulted in a valuation of $1 billion valuation in 2017 [6].

With the increase in usage of these type of applications, we also witnessed a renewed interest in the process of identification and classification of the traffic generated by these type of technologies.
There are several reasons behind this: for example, from the perspective of network operators the knowledge of the source of a particular traffic flow is useful for what concerns traffic and performance monitoring, and thus can be exploited for the design of tariff policies or to provide a better QoS via different traffic differentiation strategies. In addition, knowing the type of traffic (and in what volumes) that traverses the network can help in the design and the provisioning of the network itself.
Changing the scope to that of campus or corporate networks, there is often the need to enforce specific policies (e.g. no social networks, no gaming, and so on) and in order to do that, a certain traffic flow has to be identified first. Only then one can decide to block a flow, or maybe to favour one of them over another to improve the provided QoS.

Last but not least, the correct identification of a certain flow made in real-time can prove useful for security concerns: in particular, by means of dynamic access control (like adaptive firewalls that can detect unwanted traffic or Denial of Service (DoS) attacks) or by means of NIDS (Network Intrusion Detection System) that can help in detecting suspicious activities related to security breaches.
Therefore, accurate classification of traffic flows is a process that can be of vital importance for network administrators.

After having ascertained the importance of traffic classification, several studies have been conducted in this direction to discover the best strategy in terms of performance and accuracy.
The choice of the correct strategy to employ is strictly related to the field of application: in fact, each specific methodology has it own pros and cons (for example, one can be faster than the others but less accurate). Let us say that we need to classify traffic flows for security purposes: in that case accuracy is of utmost importance, since even a small number of false negatives (i.e. undetected threats) can undermine the security of the entire system, while false positives (i.e. normal traffic detected as a security breach), even if less of a serious problem, requires an admin to carefully analyze the detected threat to avoid blocking of non-malicious traffic flows.
In other cases a small number of false positives and false negatives may be acceptable, while a faster recognition process may be preferred.

Since the Internet (and all the protocols that make it work) keeps changing and evolving over time, these classification methods had to keep up and constantly adapt. For this reason, and also for the aforementioned need for different strategies when dealing with different problems, a wide range of traffic classifiers have been designed and accurately tested. For the sake of example, many of these classifiers were based alternatively (or even on a combination of) on the inspection of well-known port numbers, on a process of deep packet inspection, on the exploitation of flow-level characteristics (like number of packets, mean number of bytes, mean inter-arrival times, and so on) and lastly on the adoption of Machine Learning algorithms. Over time some of these methods have proved to be inefficient while others have managed to successfully be applied in some practical fields.
The next chapter will explore these different methods, why some of them are no longer applicable in today's environments and which strategies are mostly being used nowadays.

## 1.1 Objectives

The goal of this work is to evaluate the design and implementation of a tool able to detect VoIP applications thanks to the analysis of the generated traffic. The main aspect of this tool is that it must be able to classify a certain traffic flow in real-time, which means that it cannot wait for entire flow to finish but has to exploit only the first few packets in order to assign it a label.
In particular, the study has been conducted on two specific VoIP applications: Skype and Zoom. The aforementioned tool must be able to detect these two applications only during a call (both audio and video), with the only requirement being having access to the header of all packets traversing the network.

The designed tool, if proven to be sufficiently accurate, can then be enhanced and deployed in a network device (such as a router) to capture the traffic and, following a detection of certain application, block or maybe privilege the labeled flow. In order to design this type of tool, preliminary studies had to be conducted: firstly to analyse the already tested methodologies in the current state of the art; secondly, since the application detection process is heavily dependent on the traffic generated by the application itself, on the traffic generated during a Skype call or a Zoom meeting, looking for a certain recurring pattern or a particular feature that is characteristic of the application under investigation that can help us in the classification process.

For a series of reasons (the main one being that nowadays payloads are often ciphered) deep packet inspection has not been considered as a viable solution. Instead, possible solutions alternatively based on Machine Learning algorithms or on the information retrieved from the header of the packets (or a combination of the two) have been taken into consideration and tested.

# Chapter 2

# Related work

## 2.1 First attempts and related issues

Traffic identification and classification has always been a topic of great interest for a variety of reasons, ranging from security to QoS and QoE guarantees. A lot of work has been produced by the research community approaching in different ways: time domain characterization and signatures based on the traffic features by means of statistical fingerprint [7] and machine learning methods, Markov-chains models of the transport and application protocol transitions [8, 9, 10, 11]. These papers, however, cope with broad classes of applications defined by the application layer protocol while we focus on application's specific traffic pattern to build input vector of a classifier, plus we exploit the adoption of ports and IP addresses value in a relative way in order to increase the accuracy of the tool.

Two of the earlier techniques that naturally came to mind were based on the port inspection searching for well-known port numbers and on deep packet inspection. Both have proved to be inefficient for different reasons. The Internet Assigned Numbers Authority (IANA) [12] assigns the well-known ports from 0-1023 while registers ports in the range from 1024-49151. Finally, ports in the range 9152-65535 are considered to be private or dynamic.
Usually, when a client sends its first packet, it contacts the server on a well-known port of a particular application while the source port is chosen amongst the dynamic ones. Since all the following packets belonging to the same TCP/UDP session keep using the same port numbers both for client and server, one could think to map the server port number to one of the IANA registered ports and thus identify the application being used. Besides, port-based classification requires little computation and memory resources since it needs only the information about the port which can be easily retrieved from the header of the packets traversing the network.

Unfortunately, it is not always a viable solution for a series of reasons:

- it could happen to have client ports in the registered port range, which could result in a false positive (mistakenly classifying that traffic flow as generated by the application that is registered with that port)
- not all applications have registered ports with IANA
- server ports could be dynamically allocated when needed (e.g. the server port used for a data transfer of FTP can be dynamically negotiated via the already established connection that uses the well-known port instead)
- some applications purposedly use ports registered for other purposes to circumvent control restrictions (e.g. when a firewall is being used to block unknown or suspicious traffic, an application may use port 80 even if it does not generate HTTP traffic, because most firewalls do not block this type of traffic)
- port registrations are not always distinct and clear
- the possible use of certain technologies like NAPT (Network Address and Port Translation)

The other approach that was attempted was to inspect the payload of the packets being captured ([13, 14, 15]).
This strategy relies on a certain knowledge about the payload formats for each application under analysis looking for a recurring pattern. It has proven to be extremely accurate, especially when using it for the first packets of a flow that capture the negotiation/handshake phase of the application that is usually made by a predetermined set of packets with a certain payload format which are often quite different from application to application.
But even this methodology is not free from problems, and is thus often inapplicable to real-world traffic since:

- unlike port-based classification, this type of technique is really complex to implement and it needs a lot more of processing power and memory resources from the device on which the classification process takes place
- dealing with proprietary protocols is not always feasible, unless a previous work of reverse-engineering was made to study the behaviour of the application
- the Internet (and with it all the protocols that make it work and all the applications that rely on it to function) are constantly changing; this means that this payload-based classification process must be constantly kept up-to-date with patches and updates as sooner as possible

- direct inspection of the content of packets may cause privacy-related issues
- nowadays the payload of the packets traversing the network often happens to be encrypted

A new trend then arose: classification by exploitation of summarized flow information, such as duration, number of bytes, number of packets, inter-arrival times and so on. This new approach has another limit: it needs the entire flow and thus has to wait for it to finish before assigning it to an application.

## 2.2 Machine Learning as a solution to traffic identification

In [16] a machine learning system for classification of the traffic was adopted, adding a simple but really effective parameter in the classification process: the number of TCP flows belonging to the same HTTP video session, since streaming services tend to use more than one connection to deliver the video content. This is a key feature of VoIP and video applications which has been used also in our project to group together UDP sessions belonging to a certain VoIP application.
Much work on this field has focused on ML strategies for the detection of malware traffic [17, 18, 19]. In [20] they detect and counteract a low-frequency DDOS attack by exploiting the intrinsic periodicity of such malicious application. Their results reveal that normal TCP flows can be segregated from malicious flows according to energy distribution properties.
Of particular interest are the considerations exposed in [17, 19] where they highlight the primary reasons for slow concretization on the adoption of the classification methods for malicious traffic: inaccurate ground truth and a highly non-stationary data distribution. These two key pitfalls are indeed true also in our context of VoIP traffic identification since the encrypted application data pattern may change according to the will of the backing Company. This implies a periodic revision of the assumptions, models and algorithms adopted for the identification based on the observations.

Another important paper (since our work took inspiration from it) is [21], where, instead of other solutions where the entire flow is needed in order to label it, a new technique is proposed, based on early classification (i.e. only the first packets of the flow are used for the application recognition process), which is indispensable if we want to perform automatic blocking or filtering of specific applications in real time.

In particular, the size of the first P packets of the TCP flow (NB: TCP control packets (SYN, ACKs, etc.) are discarded from the analysis) are used to identify the application since these packets usually capture the application's negotiation

phase, which most of the time is a pre-defined sequence of messages. Since we only need a little piece of information which is easily retrieved from the header of the packets this approach can be considered to be quite efficient both memory-wise and performance-wise.

It is needed to have:

- offline access to flows generated by a certain number of known applications for training purposes
- access to both directions of the TCP connection that we want to classify
- online access to header of all packets for classification purposes

Two different datasets (collected on the same link but months apart) were used, one for the offline learning phase while the other for testing the accuracy of resulting classifier.
In the learning phase, TCP flows are first extracted from the training set, then each one of them is associated to a spatial representation: a point in a P-dimensional space where the p-th coordinate is the size of p-th packet in the flow. Packets sent by the server have a negative coordinate.
Then, instead of using a "traditional" supervised classifier, the authors of [21] propose to use a solution based on unsupervised clustering since a single application can exhibit multiple behaviors which should be modeled separately (in fact, unsupervised clustering, instead of having one label per application, relies on unlabeled data samples (which in this case are the TCP flows, represented by the sizes of the first P packets) to find clusters in the training set). The chosen clustering algorithm is K-Means algorithm. For each flow the Euclidean distance between its spatial representation and the center of each predefined cluster is computed, to find and choose the nearest one.
At the end of this phase we have the center of each cluster and the set of applications belonging to them. It was found that the best separation of applications among clusters was observed with P=5 packets and that for that value of P, 50 clusters was the best trade-off between behavior separation and complexity.

The resulting classifier consists of four different modules:

- Packet analyzer: after having filtered out control traffic, it extracts the 5-tuple (protocol, IPs and ports) and the packet size from the header of each packet, and stores the size of every packet in both directions of the connection until it has the first 5
- Flow converter: maps the new flow (which is an array of 5 sizes) to the spatial representation
- Cluster assigner: finds the nearest cluster to the point representing the flow

- Application identifier: selects which application of the chosen cluster is the most likely to have generated that traffic, exploiting a simple heuristic (i.e. the most common application in the cluster)

More than 80% of flows of almost all of the applications under analysis were correctly identified. The accuracy, that is lowered by certain applications that are not the dominant one in their cluster, can be improved by considering extra information or by using a better heuristic.

Possible problems and challenges:

- packets can arrive out of order or multiple times: fortunately, since only the 5 packets are needed to arrive in order and that is very likely to happen
- especially on high-speed links, sniffers could apply packet sampling which would result in some of the packets missing
- not always two distinct applications have two clearly distinct negotiation phases in terms of sizes of the first 5 packets
- even if in a small number, false matches are possible: this, unfortunately, prevents the use of this type of techniques to automatically filter or block flows since it could block traffic that it should instead be allowed; nonetheless, it is still useful to have a first process filtering and then have network administrators manually inspect suspicious flows for further understanding

Finally, since our study is conducted employing Zoom and Skype audio/video sessions as use-cases, we refer to [22] for the first detailed investigation on the identification of Skype sessions. The article is dated (2007) and the considerations based on the observations made to build the model for the identification are no more valid. Our work is a little contribution towards the characterization of those two popular VoIP apps since the model was built based on more recent observations.

# Chapter 3

# Traffic analysis

## 3.1 Preliminary studies

Before diving into the design of the detection tool, a series of studies on the nature of the traffic generated by the two applications under analysis have been conducted. In particular, about the traffic generated during a Skype call (both caller side and receiver side) and a Zoom meeting (both during the creation of a meeting and while joining to one).
This preliminary phase of study is, in fact, fundamental if we want to reach a good degree of accuracy, since it allows us to discover certain patterns or particular features of the traffic that can help discern one application from another.

First of all, only the UDP traffic was considered for this purpose (the work can subsequently be extended to include TCP packets into the analysis). In particular, it was first studied the number of UDP sessions generated by the two applications during the call/meeting and the number of different IP addresses and of different UDP ports used by these sessions. In order to identify the sessions a sniffing tool was used during a Skype call or a Zoom meeting. Then, the captured traffic was inspected, after discarding uninteresting sessions (non-UDP packets, DNS and MDNS packets, NTP packets, etc.) because not related to the application or because they do not represent a distinctive trait of it (e.g. some DNS packets are probably generated because of the call but they are not as distinctive as, let's say, a control session of the call/meeting).
Other studied features were the number of packets per session generated during the first $N_{sec}$ seconds and the variation of throughput per session in the first 15 seconds. "The first $N_{sec}$ seconds" are to be intended per session, in that each session has its own countdown timer that starts when the first packet arrives. Several values of $N_{sec}$ were tested, as long as they were not too high. In fact, taking inspiration from [21] it was taken the decision to exploit only the information generated during

the early phase of the application call, both because the purpose of this work is early recognition and because the early phase often captures the negotiation phase which is usually the most distinctive one.
Unlike the number of packets, the variation of the throughput allows us to see a different piece of information (the number of generated bytes instead of the number of generated packets) and to see the evolution of the traffic behaviour during the first 15 seconds instead of looking at a "snapshot" (the number of packets generated during the first $N_{sec}$ seconds is a value, while the variation of throughput is a series of values).
Other tests were performed to gather new information about the sessions. In particular, the aforementioned features were studied under different conditions, that is with audio and video alternatively turned off and on. From the gathered data some assumptions about which session carries what can be made (e.g. if video and audio travel in the same UDP session or not, if we have control sessions, etc.).
For the traffic analysis a series of tools were used:

- Wireshark [23]
- tcpdump [24]
- an ad-hoc Python script developed specifically to sniff traffic, divide packets per session and extract or compute certain features

Wireshark is particularly useful because it also shows the application layer protocol (when deductible from the used ports or from a particolar format of the payload, if it is not ciphered).

## 3.2 Traffic generated by Zoom

Let us focus first on Zoom and, as we already said, only on the UDP traffic.
First of all, to make a Zoom meeting one must first create the meeting, and only then other users can join by typing its URL and password. But one particular thing distinguishes the two cases (i.e. creation of a meeting and join to one): after a user types in URL and password he has to wait for the host of the meeting (i.e. the user who created it) to accept him. This causes the traffic generated in the two cases to be slightly different.
The figure 3.1 shows the UDP sessions generated during the first few seconds of the creation of a Zoom meeting.
In particular, three sessions were identified. These sessions are almost contemporary (during the experiments it was found that they were at most 0.5 seconds apart). The endpoints of these three sessions turn out to be always the same two: the host that is creating the meeting and a server.

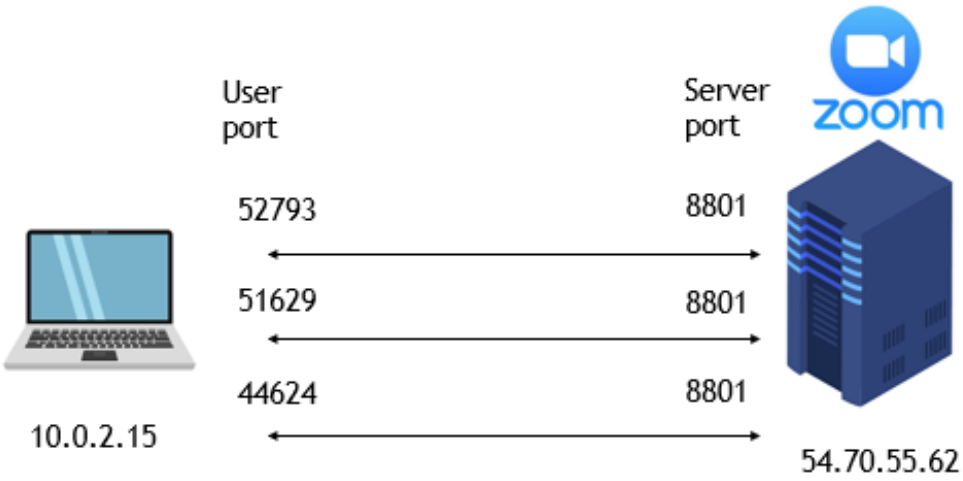


**Figure 3.1:** Creation of a Zoom meeting: UDP sessions

For what concerns the UDP ports used, the user one always changes from one session to another while the server one remains the same, not only for the three sessions of the same instance of a meeting but even for two different instances. In particular, the UDP port used by the server is always 8801.
This regularity in the characteristics of the sessions obviously proves to be very useful when designing the detection tool since these pieces of information can be exploited to our advantage. In conclusion, during the creation of a Zoom meeting we have the generation of three UDP sessions where the number of different IP addresses is two and the number of different UDP ports is four.

The figure 3.2, instead, shows the UDP sessions generated when trying to join a Zoom meeting.
Here the two phases are pretty clear. In fact, we notice how this time the number of UDP generated sessions is six instead of three.
In particular, during the first phase (after the user has typed in the URL of the meeting and the corresponding password) we witness the generation of three UDP sessions. These sessions have the same characteristics of the three sessions generated during the creation of the meeting, at least for what concerns IP addresses and ports: they are all established between the same two hosts (the user trying to join and a server), the user port keeps changing from session to session while the server port is always the same (8801).
This situation will remain the same indefinitely, unless the host of the meeting

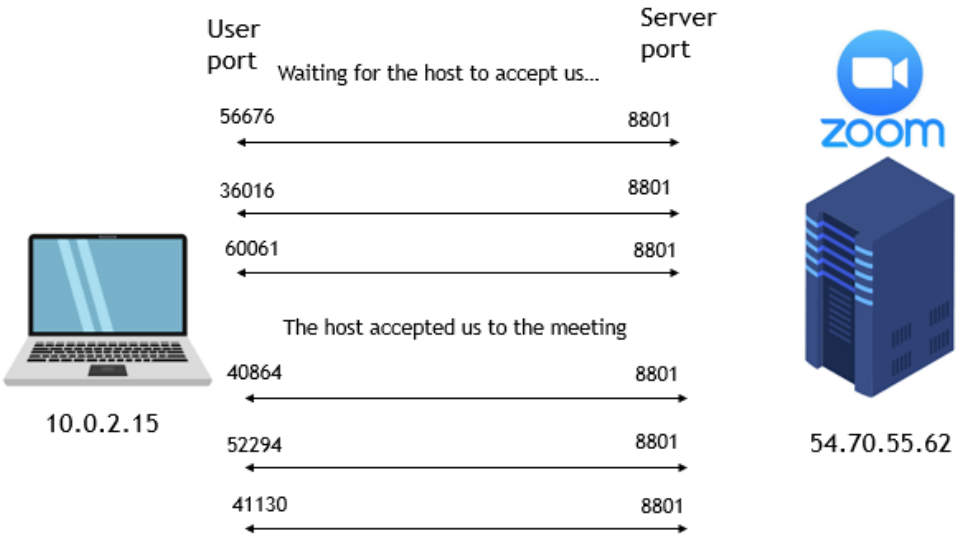


**Figure 3.2:** Join to a Zoom meeting: UDP sessions

decides to accept the user trying to join. In that case, the first three sessions will cease to generate packets while three new UDP sessions will appear, having the exact same characteristics as regards IP addresses and ports. These last three sessions will keep generating packets during the actual meeting.
In conclusion, in the case of join to a Zoom meeting we have a total of six UDP sessions, a number of two different IP addresses (since the last three UDP sessions during the meeting use the same server of the first three sessions during the "waiting phase") and a number of seven different UDP ports (six different user ports and only one server port (8801)).
Even if the behaviour of the application is different in the two cases, we can still identify a certain regular pattern in it.
Wireshark in this case does not give us any valuable information (the "protocol" field just tells us "UDP" while the payload is ciphered, both for each of the six sessions generated during join and for each of the three sessions generated during creation of the meeting).

As it was already mentioned, further studies on the sessions were conducted. The graphs shown in the figure 3.3 display the number of generated packets per session in the case of join to a Zoom meeting under different conditions.
"A" stands for "Audio" and "V" stands for "Video", while 0 and 1 mean respectively

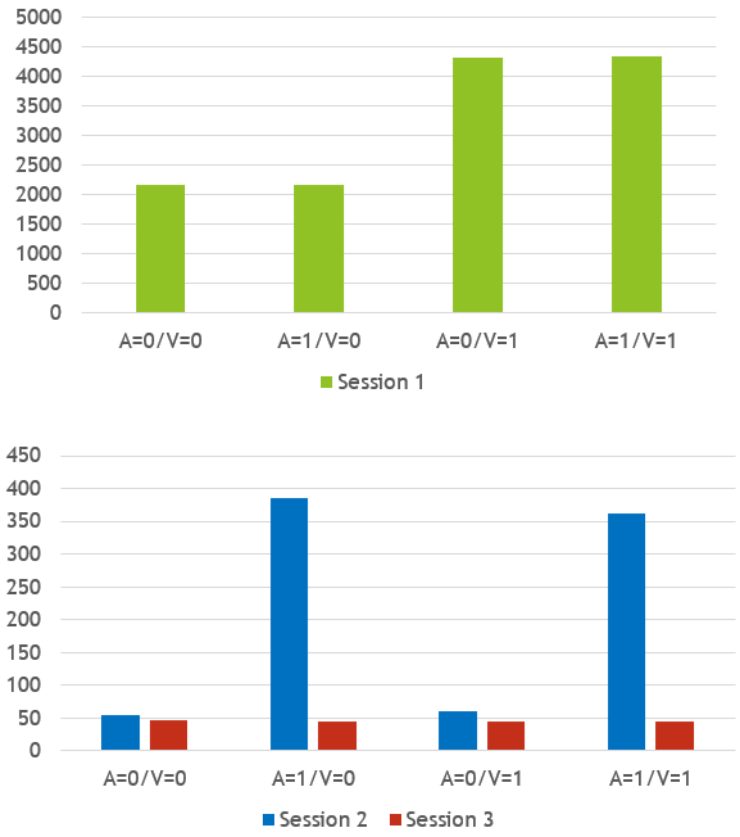


**Figure 3.3:** Join to a Zoom meeting: number of generated packets per session in the first 15 seconds under different conditions for audio and video

off and on (both receiver and caller side).
Note that, for the sake of readability, the data about session 1 are shown in a different graph since it generates a much larger number of packets with respect to the other two sessions.
Besides, the graphs show the information about only the last three sessions generated when joining to a meeting (i.e. the ones during the actual meeting) since the number of packets of the first three sessions (i.e. the ones during the waiting phase) is less relevant because it is really variable and strongly dependent on when the host of the meeting accepts the user trying to join.
A series of interesting information can be deducted from these graphs:

- the first session generates a much larger number of packets than the other two.
- the number of packets generated by the first session does not change if the

audio is turned on or off, while it practically doubles its value (approximately from 2000 to 4000 packets) if the video is turned on.

- the number of packets generated by the second session basically behaves opposite of the number of packets of the first session: in fact, this time this value is indifferent to the state of the video, while it drastically rises up (approximately from 50 to 350 packets) if the audio is turned on.

- lastly, the number of packets generated by the third session is not affected by the state of audio and video and, with respect to these two variables, remains pretty much fixed to a value of approximately 50.

Based on this data, it can be assumed that realistically the session 1 transports the video segments, the session 2 transports the audio segments while the session 3 could transport control data.
But strangely enough, even with the video turned off, the session 1 still generates a large amount of packets (approximately 2000) and only doubles its value if instead the video is on. It is true that a similar thing happens for the session 2 but in that case the number of generated packets with the audio turned off is just 50 (which can be explained as representing control packets) and the value with the audio on is more than seven times bigger.
A thing not to ignore is that there is always the chance that audio and video could also exploit TCP sessions but since the analysis is restricted to UDP packets it cannot be ascertained.

To inspect the sessions from another point of view, let us now look at the figure 3.4 which shows the variation of throughput per session in the first 15 seconds of the last three sessions generated when joining a Zoom meeting.
In particular, the value relative to the axis X is the number of generated bytes by the session from the X-1 to the X second.
The information inferred from these graphs basically reinforces the previously made assumptions:

- Session 1: under different conditions of audio we have basically the same graph while there is a noticeable difference if the call is with or without video. If we have no video, after the first 8 seconds when we still have an average throughput of 2-3 KB/s, the session practically stops generating bytes. On the other hand, if the meeting also hosts the video, after the first 8 seconds in which we even have peaks of 4.5 KB/s, the session settles down and keeps generating bytes throughout the meeting with an average of 2-2.5 KB/s. This reinforces the fact that this sessions could be the one transporting the video segments.

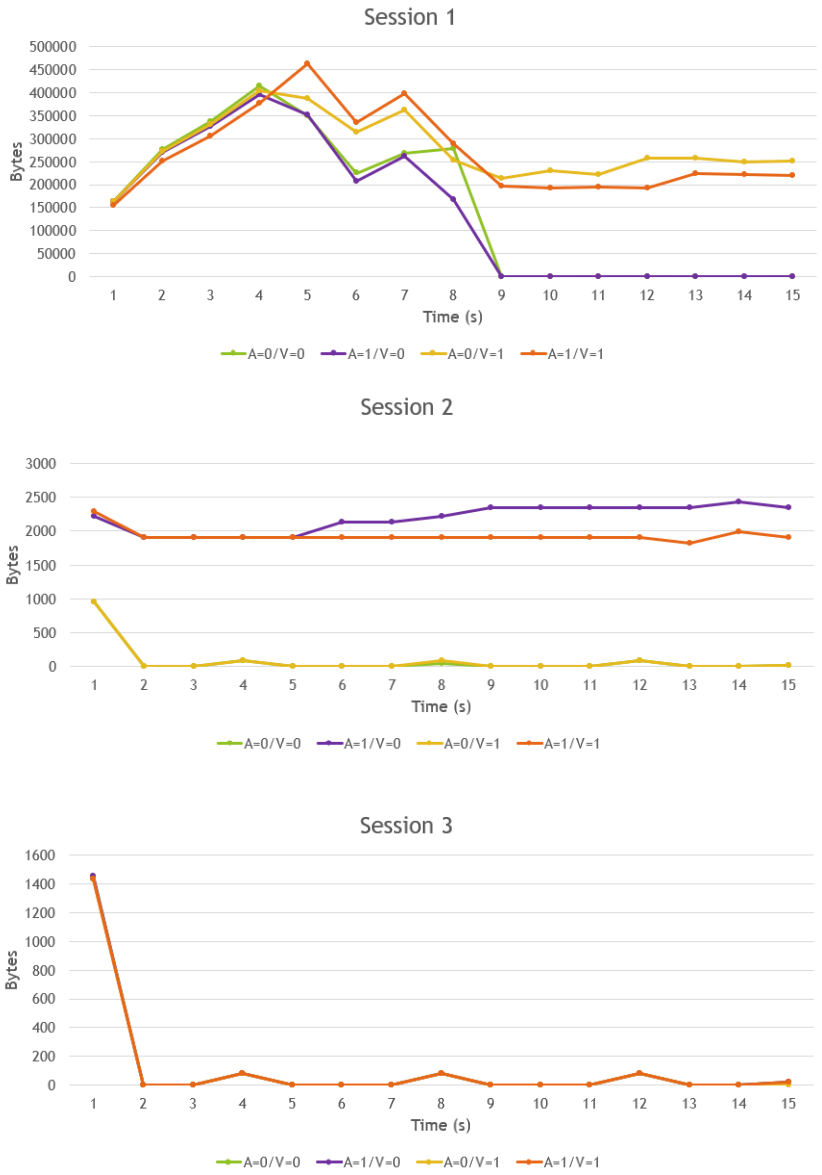


**Figure 3.4:** Join to a Zoom meeting: variation of the throughput per session in the first 15 seconds under different conditions for audio and video

- Session 2: in the case of meeting without audio, after an initial peak of about 1000 bytes, we have only the periodic generation of about 80 bytes. In the case of meeting with audio the session average throughput is 2 KB/s. That is why it could be the session transporting the audio segments.
- Session 3: under different conditions of audio and video the variation of the

throughput follows the same pattern which is a higher peak in the first second of approximately 1400 bytes and smaller periodic peaks of about 80 packets. It could potentially be a control session.

It must be said that these values of throughput are to take with a grain of salt since they also depend on the environment in which the experiments have taken place.

Lastly, let's give a look at the number of generated packets per session in the first N_SEC for different values of N_SEC.
The figure 3.5 shows the number of packets per session of the last three sessions generated during the first N_SEC of join of a Zoom meeting.
Both the session 1 and the session 2 generate packets with a more or less constant rate while we can see that practically almost the entirety of packets generated by the session 3 in the first 15 seconds are in fact already generated in the first 5 seconds.
Finally, the figure 3.6 let us see the number of packets per session of the three sessions generated during the first N_SEC of creation of a Zoom meeting.
Here we notice how the session 1 still generates a good amount of packets, particularly in the first five seconds (1500 packets) while the other two sessions generate just some dozens of packets.
The interesting thing to notice is that, this time, not only the session 3 but even the other two sessions after a few seconds stop generating packets (or generate really few packets). This is especially visible in the graph about session 1 since from the 10th to the 15th second we do not witness a marked difference in the number of generated packets. This is probably due to the fact that during the first 15 seconds of the creation of a meeting the other user has not joined the meeting yet, so there is no exchange of audio and video segments (the few generated packets after the first few seconds are probably control and keep-alive packets).

## 3.3 Traffic generated by Skype

Now let's focus on the UDP traffic generated during a Skype videocall. Similarly to the Zoom case we have two different "roles": in this case, they are the caller and the receiver, or in other words the user that tries to establish the videocall and the user who can respond to it (or not). The traffic generated in the two sides is slightly different.
First of all, unlike for Zoom, the number of UDP generated sessions was not always the same but it was proven to be slightly variable. For this reason, just a subset of these sessions was analysed, in particular only the set of sessions that seems to be generated every time.

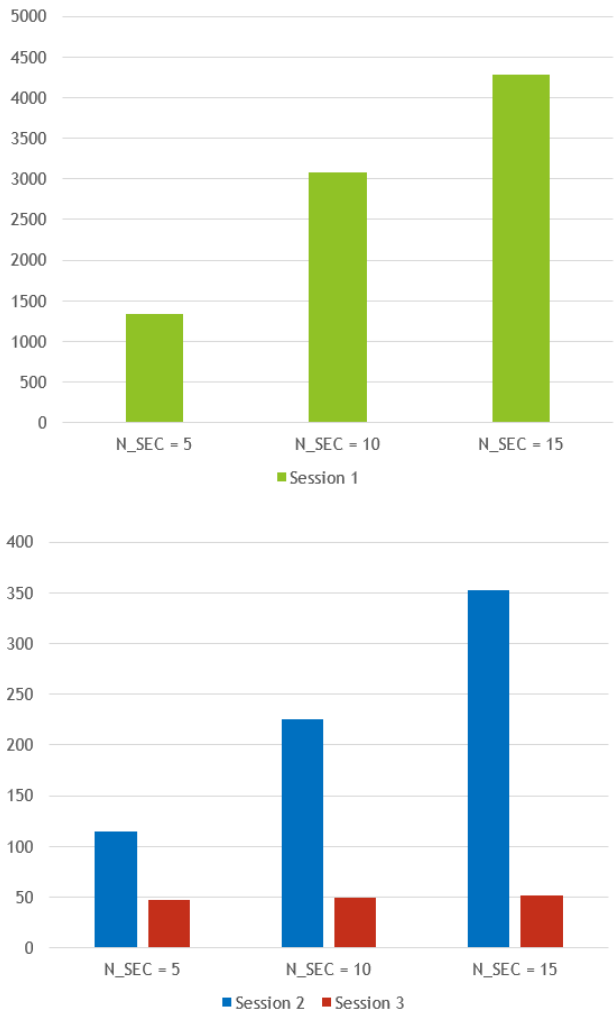


**Figure 3.5:** Join to a Zoom meeting: number of generated packets per session in the first N_SEC

The figure 3.7 shows the UDP sessions generated during the first few seconds of the reception of a Skype videocall (i.e. video and audio), starting from the ring of the call.

When the call starts ringing, four UDP sessions immediately start generating packets. These sessions, during the experiments, were usually almost contemporary, 0,6-0,7 seconds apart at most. After a few seconds, if the call is not picked up and it keeps ringing, no new packets are generated. If, instead, the call is picked up, two of the four sessions start generating packets again.

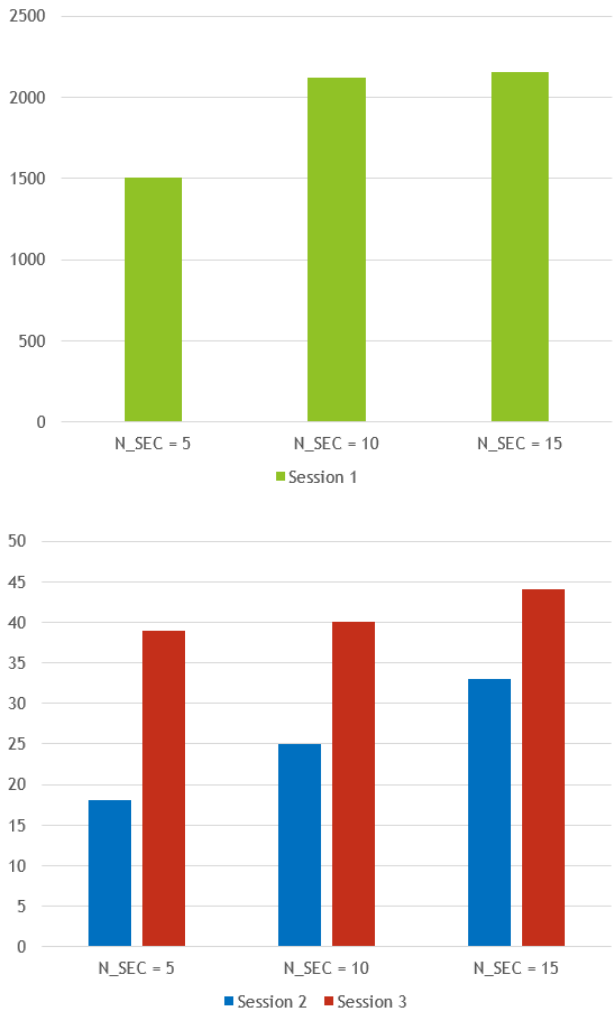


**Figure 3.6:** Creation of a Zoom meeting: number of generated packets per session in the first N_SEC

Taking a closer look at the UDP sessions:

- two sessions have basically the same behaviour: their endpoints are the host receiving the call and a server, they generate really few packets (usually 2) and only during the ringing of the call and the application layer protocol inferred by Wireshark is STUN.
  The STUN (Session Travel Utilities for NAT) protocol, as specified in the RFC 5389 [25], is a useful tool for dealing with NAT traversal, specifically to determine the IP address and port allocated to an endpoint by a NAT.

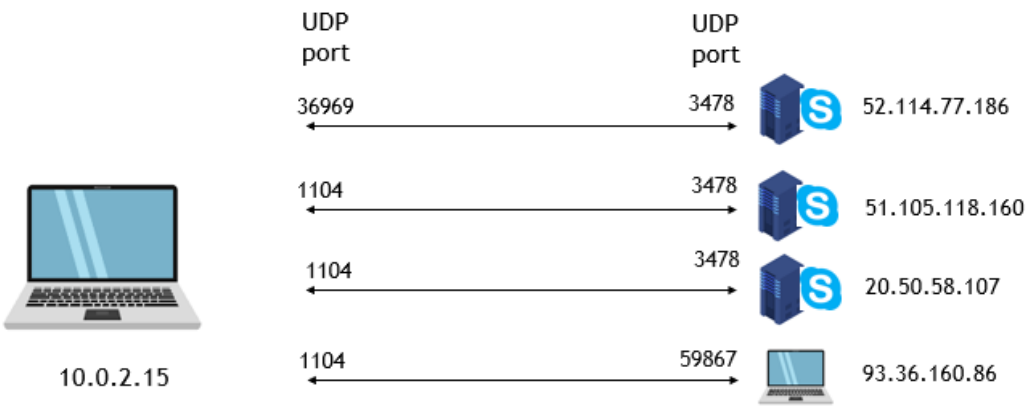


**Figure 3.7:** One of the possible cases of generated UDP sessions during a Skype videocall (receiver side)

- the third session is similar to the first two in that it has similar endpoints (the host receiving the call and a server) but, while it could seem that the application layer protocol is still STUN since the port used by the server is the same of the other two sessions (port 3478) which is a registered port for the STUN protocol, we cannot be sure given that the payload is ciphered.
  Besides, it differs for what concerns the generation of the packets: it starts by generating about ten packets during the ringing of the call and it does not stop there since if and when the call is picked up it starts generating packets again.
  In particular, it generates a dozen of packets the moments immediately after the call is picked up and then remains active throughout the whole videocall by generating some periodic packets every few seconds.

- the last session is pretty different from the other three. In fact, first of all is not a client-server session since the two endpoints this time are the host receiving and the host making the videocall. It starts by generating approximately six packets concurrently with the other three sessions.
  The application layer protocol inferred by Wireshark for these packets is STUN. After the call is picked up it starts generating a large amount of packets which this time are ciphered (and Wireshark can no longer assume anything about the protocol) until the the call is ended by one of the two endpoints. During the videocall, among these many ciphered packets we also have some sporadic STUN packets.

In conclusion, when receiving a Skype videocall we witness the generation of a total

of four UDP sessions. Two of them (i.e. two of the ones going from the receiver of the call to the servers) are STUN sessions, while for the other client-server session and the other session (i.e. the one between the two users making the call) we have no info about the application layer protocol.
The number of different IP addresses and of different UDP ports unfortunately is not fixed from call to call (unlike Zoom) since two of the three client-server sessions could share the same server and since there is no regularity in the value of the ports used by the host receiving the call (they could all be different for all the four sessions, or maybe two of the sessions could share the same port, etc.). That is why the figure 3.7 is just an example of a possible case.
Nonetheless, we can still find some pieces of information about the sessions which are instead fixed and can then be exploited by the recognition process.
In particular, the three client-server sessions share the same server port, both in the same call and from call to call. That port, in fact, is always equal to 3478, which happens to be in the range of the registered ports since it is used by the STUN protocol. Since the port used by the host making the call is chosen randomly, the number of different ports on the "opposite" side of the user receiving the call is two (the random port of the other user and the 3478 server port).
So, even if the behaviour of Skype is not as regular as the one of Zoom we can still identify a subset of the features of the sessions which are fixed.

In the figure 3.8 are depicted the UDP sessions generated during the first few seconds when making a Skype videocall (i.e. video and audio), starting from the ring of the call.
This time, when the call starts ringing we witness the generation of six UDP sessions. Exactly like for the receiver side, these sessions generate a certain number of packets and then stop until the call is picked up. Only two of the six sessions resume the generation of packets during the videocall. Even for the caller side the sessions were usually 0,6-0,7 seconds apart at most.
Taking a closer look at the UDP sessions:

- four of the five client-server sessions generate really few packets (not more than 6) only during the ringing of the call. The application layer protocol used by these sessions is STUN.

- the last client-server session is ciphered and generates a dozen of packets during ringing, another dozen immediately after the call is picked up by the host on the other side and keeps generating periodic packets during the whole videocall.
- the last session, which is the one between the two hosts making the call, has obviously the same behaviour as the one between the two hosts on the receiver

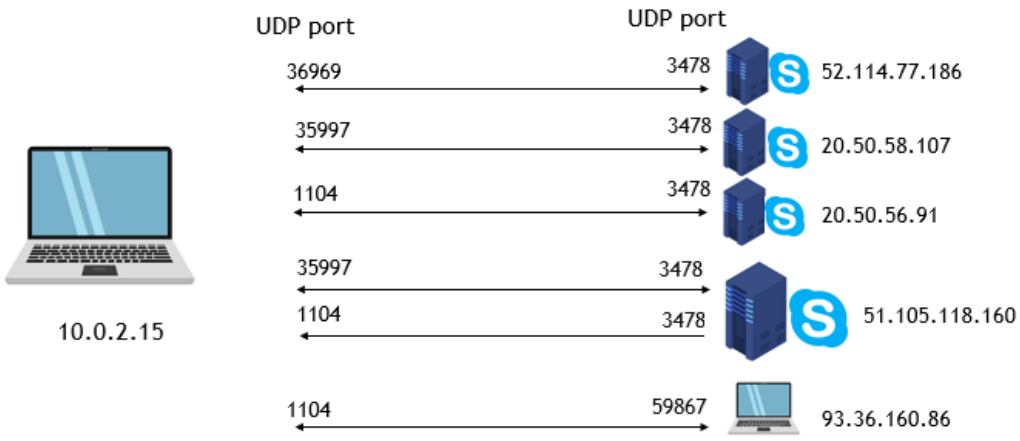


**Figure 3.8:** One of the possible cases of generated UDP sessions during a Skype videocall (caller side)

side: it generates approximately six STUN packets during ringing and starts generating a large amount of UDP ciphered packets during the actual videocall (and among them some occasional STUN packet).

In conclusion, when receiving a Skype videocall, we witness the generation of a total of six UDP sessions. Four client-server sessions use the STUN protocol, while for the other two sessions we have no hints about the application layer protocol. Even in this case, the figure 3.7 is just an example since the number of different IP addresses and of different UDP ports is not fixed from call to call: the client-server sessions could share the same server and the UDP ports used by the host that is making the call could be repeated from one session to another with no regular pattern.
But if we focus only on the number of different ports on the "opposite" side of the user making the call that number is fixed and equal to two since all client-server sessions use the same port (3478).

Lastly if, instead of a videocall, an only-audio call is established, there is no difference in the number of the UDP generated sessions and in the characteristics of the number of IP addresses and UDP ports used while, of course, there is a difference in other features like the number of generated packets or the variation of the throughput.

Let us now look at the computed features.
The graphs shown in the figure 3.9 display the number of generated packets per

session in the case of reception of a Skype call (since in the case of making a Skype call we have just two more sessions that basically act like other two sessions already present during the reception) under different conditions.
The meaning of "A" and "V" are respectively "Audio" and "Video" while 0 and 1

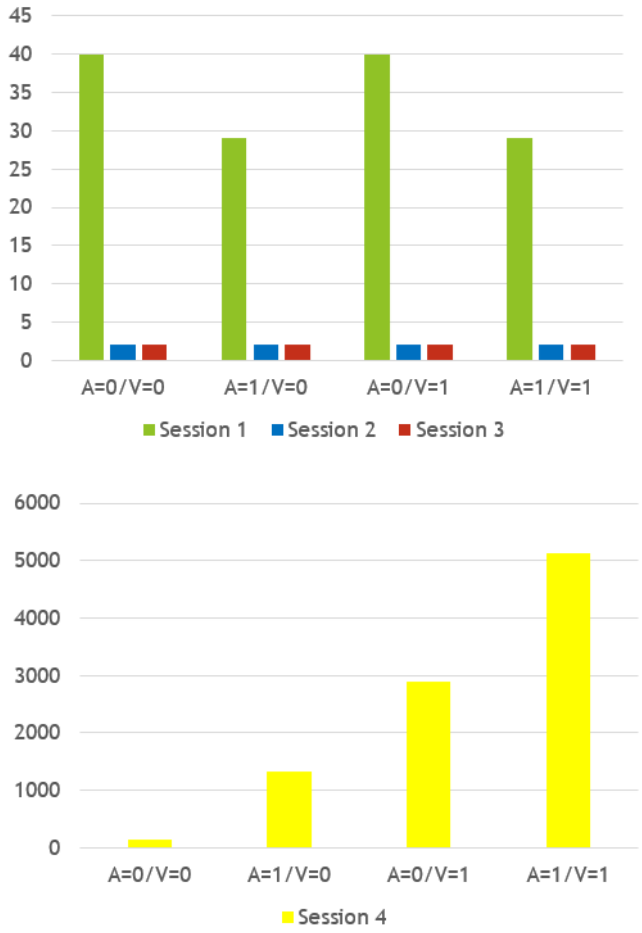


**Figure 3.9:** Reception of a Skype call: number of generated packets per session in the first 15 seconds under different conditions for audio and video

mean respectively off and on (on both sides of the call).
Note that, for the sake of readability, the data about session 4 are shown in a different figure since it generates a much larger number of packets.
The pieces of information that can be deducted from these graphs are:

- both the session 2 and the session 3 generate very few packets in the first 15 seconds (approximately 2) and this number is indifferent to the conditions of audio and video.

- the number of packets generated by the session 1 is a few dozens and it only slightly depends on the condition of audio. Strangely enough it generates approximately 10 packets more but when the audio is turned off. Actually, it probably depends on how the experiments were conducted since the audio was not turned off since the beginning of the call but immediately after the call was picked up so that 10 packets are probably control packets.
- lastly, the session 4 is dependent on both audio and video. If both are turned off, very few packets (about a hundred) are generated in the first 15 seconds. Otherwise, we witness a sharp increase in the number of generated packets which is more visible if the video is turned on (which means that the video causes the generation of more packets). In fact, with only the audio turned on we have almost 1500 packets while with only the video turned on the number rises up to 3000. Lastly, with both of them the number is even greater than the sum of the two, since it reaches peaks of 5000 packets.

Based on this data, it can be realistically assumed that the session 1, 2 and 3 transport control packets, while the session 4 is the only one that could transport audio or video (and maybe both) segments. However, we cannot be a hundred percent sure that the session 4 transports both audio and video segments since even TCP sessions could be exploited for this purpose but they were not analysed in this work.

Let's now switch point of view: the figure 3.10 shows the variation of throughput per session in the first 15 seconds of the sessions generated during the reception of Skype call.
The information that can be inferred from these graphs are:

- the session 1 throughput seems to not be dependent from the condition of audio and video. It reaches peaks of 5 KB/s in the first 4-5 seconds after which it stops generating bytes.
- both the session 2 and the session 3 are not dependent from the condition of audio and video and generate really few bytes only in the first second (almost 300 bytes for the sessions 2 and about 800 bytes for the session 3) after which they stop generating bytes.
- the session 4 throughput, instead, is strongly dependent on the state of audio and video. In fact, when both are off, the average throughput during the first 15 seconds is 1-2 KB/s while if we turn on the audio only it rises up to 9-10 KB/s, with peaks of 15 KB/s. Finally, a sharp increase is visible if also the video is turned on (and this reinforces the fact that this session could transport video and audio segments). In fact, in that case the average throughput is 450 KB/s, with peaks of more than 600 KB/s.

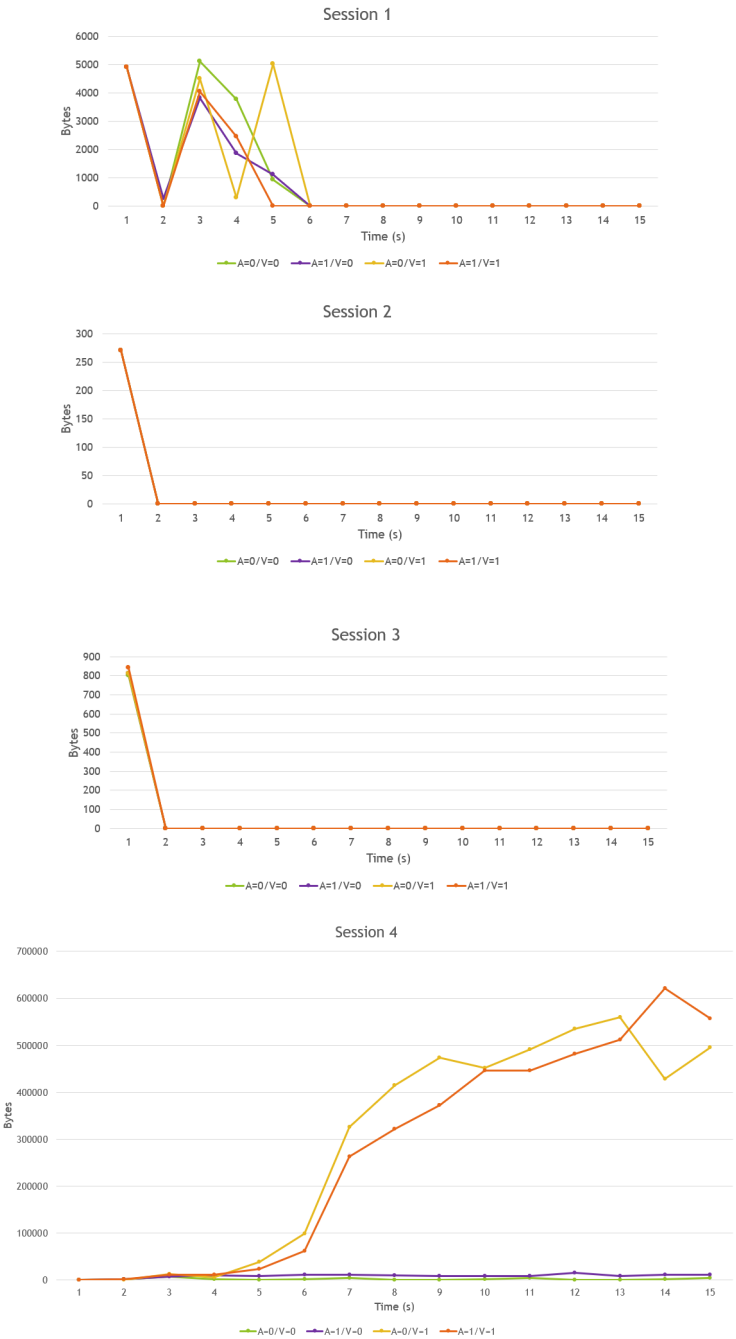


**Figure 3.10:** Reception of a Skype call: variation of the throughput per session in the first 15 seconds under different conditions for audio and video

It is worth pointing out that, like it was already mentioned in the Zoom section, these values of throughput are to take with a grain of salt since they also depend on the environment in which the experiments have taken place and they are really variable, especially for the session that potentially contains the audio and video segments (session 4).

To conclude this chapter, let's now give a look at how the number of generated packets changes over time. In particular, the figure 3.11 shows the number of generated packets per session in the first N_SEC when receiving a Skype videocall (i.e. video and audio).
We can notice how the number of packets generated by the first three sessions does not change from the 5th to the 15th second and this reinforces the fact that they cannot transport audio or video segments since otherwise they would keep generating packets even after the 5th second. Finally, the session 4 keeps generating packets over time, even if not at a constant rate (at least in the analysed range of time) since the variations of packets every 5 seconds are respectively from 0 to 500, from 500 to 2500 and from 2500 to 5000.
Finally, the figure 3.12 shows the same type of information but for the sessions generated when making a Skype videocall (i.e. caller side).
We notice how four of the six sessions behave almost exactly like the four sessions generated on the receiver side (in particular the session 1, 2, 3 and 6).
For what concerns the two extra sessions, one of them is practically identical to the session 2 and the session 3 while the other one generates really few packets (5-10) and it seems to keep generating them (at least until the 15th second).
Besides, the session 6 generates about 500 packets more than the corresponding session 4 on the receiver side, but apart from that the behaviour is similar (it keeps generating packets even after the first few seconds).

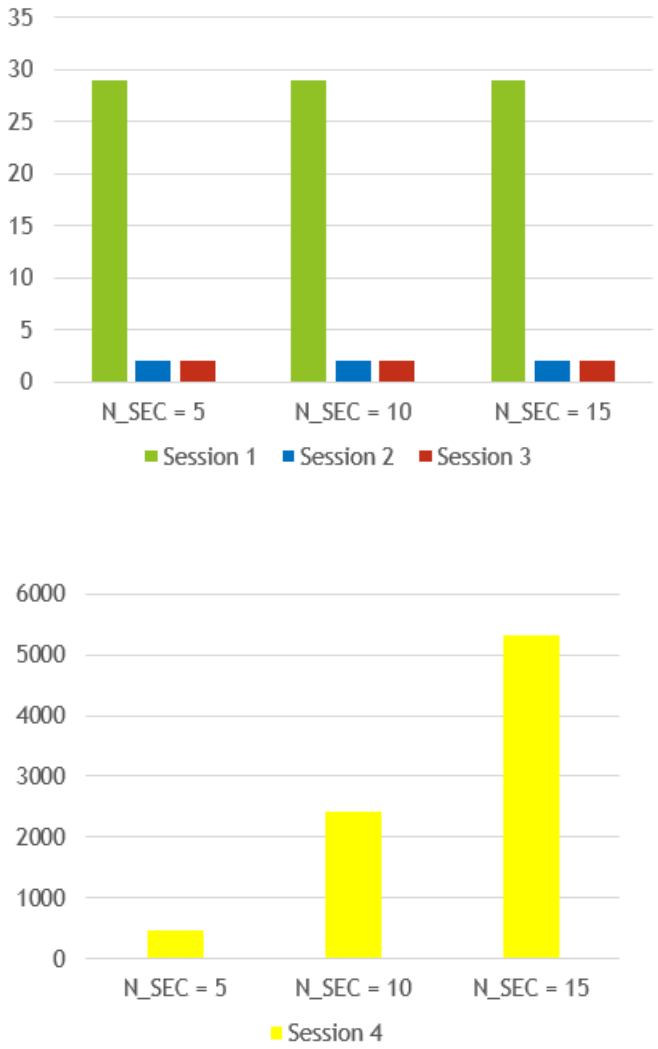


**Figure 3.11:** Reception of a Skype videocall (video+audio): number of generated packets per session in the first N_SEC

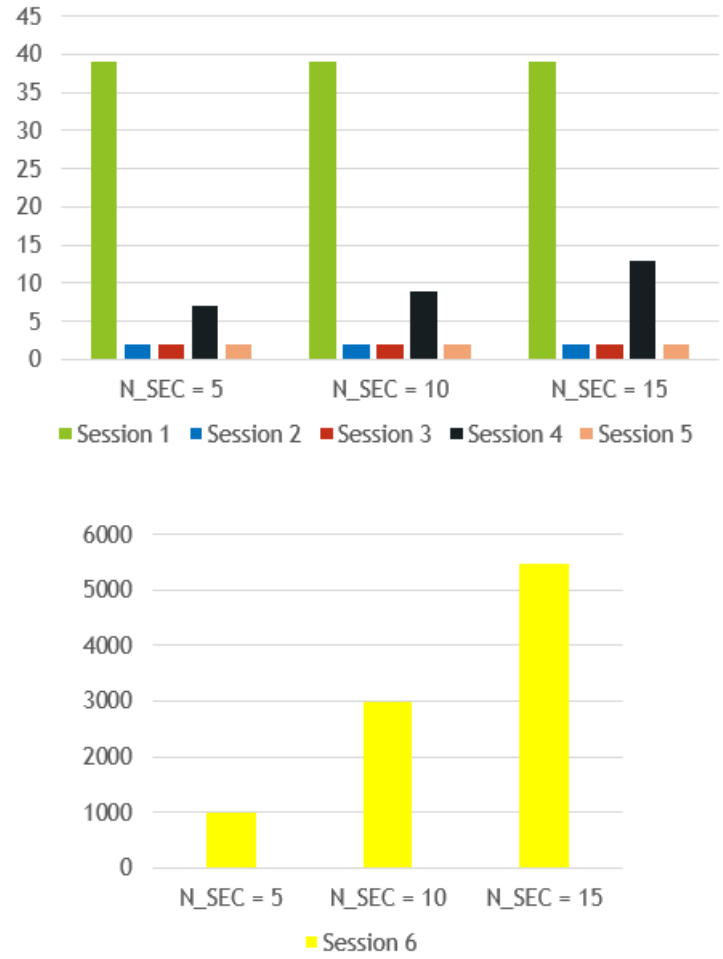


**Figure 3.12:** Making of a Skype videocall (video+audio): number of generated packets per session in the first N_SEC

# Chapter 4

# The detection tool

## 4.1 First approach

The previous chapter is fundamental for the design and development of the so called macro-classifier. In fact, since the recognition process exploits information about the type of traffic generated by the two applications under analysis, only after having studied the traffic and having found out which are the most characterizing features it is possible to define the modules of the macro-classifier.
First of all, a number of issues have emerged during the analysis:

- more than one UDP session per application: usually an application generates more than one session for its functioning. For example, when considering a VoIP application there could be a session for the video, another one for the audio and others for control messages.
  Unfortunately, even considering only the UDP sessions, this is also our case since both Skype and Zoom work with multiple UDP sessions. The consequence of this is that we cannot classify a single session as Zoom or Skype but we have to reason at group of sessions level.

- recognition of an application VS rest of the world: if we want to use a Machine Learning classifier for the purpose, we must consider that the fact that it knows only what it has been trained on. In fact, it works well when, given a certain sample whose class is unknown, it needs to apply a class label to it, chosen amongst a finite number of other labels. For example, a digit recognition classifier knows that the possible labels are only the ten digits.
  In our case, the two classes would be the application to recognize and the "rest of the world" and we would need enough samples to characterize the application and enough samples to characterize at least all the other possible applications that generate UDP traffic.

- variability of traffic: in general, the Internet and all the applications that work with it do not have a defined and constant behaviour, starting from the number of bytes and the number of packets that they generate.
  Apart from that, for Skype, we have seen also a variability for what concerns the number of different IP addresses and the number of different UDP ports. Besides, the user could be an additional source of variability since his behaviour is unpredictable (we do not know when the call will be answered, we do not know if the call will have both audio and video or if they will be turned off and on during the call itself).
  This variability complicates the work because we cannot easily detect a repetitive pattern that can help us identify the application.

- different behaviours on caller side and receiver side: since we want to detect the application both when calling and when receiving (creating a meeting and joining a meeting in the case of Zoom) and since in the last chapter we saw that the characteristics of the generated traffic in the two cases could be quite different, there is a risk of not detecting the application in one of the two sides if the classifier is not accurately designed.

These issues have to be addressed during the design of the detection tool if we want to achieve a good degree of accuracy.

After having studied the nature of the traffic generated by the two applications and the possible problems that might show up, a decision about which features and characteristics to exploit in the recognition process can be made.
In particular, it was decided to design a so-called "macro-classifier" (which is called like that because it is composed of a series of modules where each one of them has its own function) which has two main phases: in the first one we try to match the signature of the unknown sample to the one of the application while in the second (if the test of the first phase has been passed) a Machine Learning model is exploited. Using a combination of two different solutions could prove to be a good solution because the advantages of one method can compensate for the flaws of the other.
First of all, a "sample" to classify is in fact a group of more than one session, since both Zoom and Skype generate more than one UDP session. The "signature" of a sample is instead a particular pattern in the value of IP addresses, UDP ports and the relationships between them. For example, for Zoom, when creating a meeting, we have the generation of three UDP sessions, each one of them going towards the same server to the same port while the user port keeps changing. The fact that this behavioural pattern seems to be fixed can be used to check if an unknown sample containing the expected number of sessions (e.g. three in the case of creation of a Zoom meeting) matches the known signature of the application to detect.

In the second phase a previously trained ML model is used, and the features chosen to represent the sample are the number of packets generated by each session of the group in the first $N_{sec}$ seconds.
This requires the choice of a model and of a value for $N_{sec}$. Since there is not an absolute solution considering that each model works well if used in the right way and for the right purpose, the best thing to do would be to try with different models and with different values for $N_{sec}$ and find the best trade-off between performance and accuracy. Obviously, ideally $N_{sec}$ would be the smallest possible since that translates in a quick recognition of the application. After having chosen the model and the value for all the possible parameters, a number of samples representatives of all the classes must be captured and used to train the model.

## 4.2 Hints on the tool

The detection tool has been developed using the Python language (can later be translated into another language, like C, before being deployed in a router).
Main variables and data structures:

- N_SEC and N_SEC_SESSIONS: they represent respectively the number of seconds considered for the analysis of the number of packets per session and the maximum number of seconds that two sessions belonging to the same sample can be apart to be considered "related" (more details on this later). Their values are hardcoded into the program.

- sessions: an instance of an OrderedDict (which is a collections.dict subclass that keeps the insertion order). This data structure stores all the UDP sessions in a way that the key is the 4-tuple of the session, while in the value part we have the timestamp of the first packet of the session and the current number of packets generated by that session (this value stops being updated when N_SEC seconds have elapsed). Obviously, N_SEC is counted per session.

- clf: it represents the Machine Learning classifier. This classifier has been previously trained and persisted into a file thanks to the joblib.dump function [26]. Known the path of this file by passing it by command line, the corresponding function joblib.load allows to reconstruct the classifier which is then ready to use.

The tool works as follows: a new socket is first created and all traffic is captured. Each packet is then accurately parsed and its protocol stack is checked to discard all the packets that are not UDP over IP over Ethernet. For each packet, a certain amount of information is retrieved from the header: in particular the 4-tuple of the session it belongs to and the timestamp. If the packet is the first to arrive for

this session a new entry in the sessions dictionary is created storing the timestamp of the packet with it, while if it is not the first one only the number of packets is updated (but only if N_SEC are not passed yet). The first time N_SEC elapses for a session the macro-classifier comes into the picture.

## 4.3 The macro-classifier

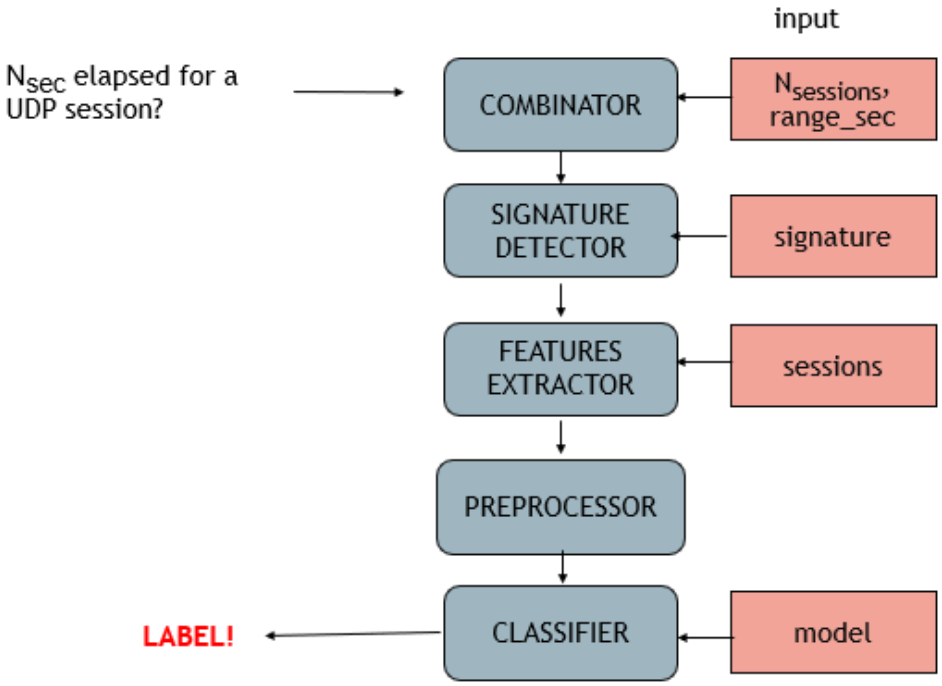


**Figure 4.1:** Structure of the "macro-classifier"

As we can see from the figure above, the macro-classifier is composed of five modules (or stages):

- Combinator: each time $N_{sec}$ seconds elapse for a UDP session (let's call it session S) the combinator receives as input $N_{sessions}$ (number of UDP sessions generated by the application to detect, whose value depends on the application itself) and range_sec (which is the maximum number of seconds that can pass from the first packet of the first session and the first packet of the last session to avoid considering as related two or more sessions that are minutes or more

apart) and selects all the possible combinations of $N_{sessions}$ sessions provided that they are at most range_sec apart.
To avoid considering a combination more than once only the combinations containing the session S are considered, while to be sure that also for the other sessions in the combination the $N_{sec}$ have passed the other sessions are chosen only amongst the one that started before the session S.

- Signature detector: this module receives the $N_{sessions}$ sessions 4-tuples (source and destination IP, source and destination port) which in fact represent the signature of the sessions.
  This signature is matched against the application signature: if there is no match the combination is discarded, while if a match is found the same $N_{sessions}$ sessions can continue to the next module.

- Features extractor: the task of this module is to build the sample that represents the $N_{sessions}$ sessions to feed to the actual ML classifier.
  This module exploits the 4-tuples of the $N_{sessions}$ sessions to look into the sessions dictionary which, among other information, contains the number of packets generated during the first $N_{sec}$. These values are then put together into an array of integers, long $N_{sessions}$, where the n-th element represents the number of packets generated during the first $N_{sec}$ for the n-th session.

- Preprocessor: this module applies all the functions of preprocessing that the ML classifier could possibly need. It always sorts the elements in the array in a certain order but in some cases it could do more than just that, like scaling or discretization of the features, potential encoding of categorical features (if present) and handling of null values (since some datasets could have some features equal to null values for some reason).
  All these functions (apart from from the sorting one) are not mandatory, and their possible use depends from the ML model chosen (since some of them will not work well if, for example, the features do not more or less look like standard normally distributed data: i.e. Gaussian with zero mean and unit variance) and from the type of features (continuous, categorical, integers, etc.).

- Classifier: the last stage receives the array long $N_{sessions}$ which has been preprocessed and feeds it to the chosen model of Machine Learning.
  Obviously, if a certain value for $N_{sec}$ has been chosen for the macro-classifier, we must train the model with the same value for that parameter otherwise the tool will not behave as expected. Besides, the chosen sorting order for the preprocessing stage must be the same chosen during the training phase of the ML model.
  Finally, the output of this module, which in fact coincides with the output of the entire macro-classifier, is the label that has been assigned to the (previously)

unknown sample which will tell us if the $N_{sessions}$ sessions have been generated by the application under consideration or not.

### 4.3.1 Zoom

As an example, let us consider Zoom first. During our studies it emerged that the number of UDP sessions generated by Zoom is equal to three in the case of creation of a meeting while is equal to six in the case of join to a meeting since we have the two phases (three sessions while waiting to be accepted by the host, three sessions during the meeting itself).
In this case, our choice for $N_{sessions}$ is facilitated by two main facts:

- in the case of join to a meeting, the six sessions are not contemporary but can be divided into two groups of three sessions (since when the host accepts the user trying to join the first three sessions cease to generate packets while the second three start to do that)

- in the case of join to a meeting, the signature of the second three sessions is practically the same to the one of the three sessions generated during the creation of the meeting (i.e. three sessions all going between only two IP addresses (the one of the user creating/joining to the meeting (let's call it side A) and an IP address of a public server (let's call it side B)) where the port relative to the side A is different in all of the three sessions while the port relative to the side B is always the same (8801))

Because of these two observations a decision was made: in the case of join to a meeting, only the last three sessions are considered. This way, there is no need to consider two possible values for $N_{sessions}$ for the same application. In particular, in the case of Zoom, $N_{sessions}$ has been placed equal to three.

In this case there could be two problems: when the tool is sniffing the traffic looking for Zoom-related UDP sessions and if there is an ongoing Zoom meeting, the actual detection happens only after that $N_{sec}$ have elapsed but since the start of the group of the last three sessions. That means that while we are waiting for the host of the meeting to accept us, the tool will not detect Zoom. This is not necessarily a problem if we want to detect the real meeting between the two users. The other problem that could arise is when (and if) the number of packets per session in the first $N_{sec}$ seconds of the first three sessions is comparable to the number of packets of the last three sessions. In fact, since the signature of the first three sessions is the same of the last ones, if the tool is active and at least $N_{sec}$ seconds pass before the host accepts the user who wants to join the meeting, there will be a double detection of Zoom even if a single instance of a meeting is being held.

Another thing to consider is that, even if the signature of the last three sessions in the case of join is the same of the three sessions in the case of creation of a meeting, we cannot say the same thing for what concerns the number of packets in the first $N_{sec}$ seconds. This consideration needs to be addressed, especially during the training of the ML model.

Going back to the macro-classifier pipeline in the case of Zoom, for range_sec a value of 1 second was chosen, since it was proven experimentally that usually the three sessions are in a range of 0.5 seconds at most (it was chosen 1 second to have a little margin of uncertainty). So, anytime $N_{sec}$ seconds elapse for a UDP session all the possible combinations of three sessions are considered, as long as they all are in a range of at most 1 second. Then, if all the three sessions share the same destination server (both same IP and same port) while the user port changes from session to session, the features array is constructed, preprocessed and given to the ML classifier.

### 4.3.2 Skype

Unlike Zoom, we cannot identify a fixed signature for what concerns IP addresses and UDP ports. In fact, as we have seen in the previous chapter, there are some features that are potentially variable and can change from one call to another (in particular, the number of different UDP ports used by the caller/receiver and the number of different IP addresses on the opposite side). This problem can be tackled simply by exploiting only the pieces of information that are constant and fixed from a call to another (i.e. the fact that we have always the same number of sessions (it changes only between calling and receiving), the fact that $N_{sessions}$-1 sessions all use the same server port (3478) and obviously the fact that on one side of all the sessions we only have one IP address (which is the address of the caller/receiver that is sniffing the traffic)).
Another problem is that the choice of a good value for $N_{sessions}$ is not easy. In fact, it was found that the number of UDP sessions generated by Skype is equal to four in the receiver side while it is equal to six in the caller side. But, unlike Zoom, we cannot cleverly select four of the six sessions in the case of calling to consider the same number of sessions in both cases since there is not a clear distinction (all six sessions are practically contemporary and logically equivalent).
Since the ML classifier must receive samples (i.e. arrays) long the same, at the current state we are not able to recognize both when making and receiving a Skype call with the same macro-classifier (or without changing a little bit the design of it). For the moment, let's focus on the recognition of a reception of a Skype call. $N_{sessions}$ would then be four, while range_sec is set equal to one.
Finally, the signature, as we already discussed, represents only the fixed information about IP addresses and UDP ports of the sessions.

## 4.4 Training of the Machine Learning classifier

These classifiers need to be trained before they can be used. "Training" a ML classifier means to capture a certain amount of traffic generated by all the applications that we want to classify. Then we need to convert this traffic to a format understandable by the trainer of the classifier, which usually is an array (the "sample" to classify) whose elements are usually numbers (the "features" of the sample).
At this point we have two sets: the set of "training" samples and the set of "labels" that contains a label for each sample. The label is also called the "class" of the sample, and in our case represents the application that generated the sample.

As we already saw in the previous sections, a series of issues needs to be addressed during the training of the ML classifier.

- multiple sessions per application: the already mentioned solution to this problem is to use as features the number of packets in the first $N_{sec}$ seconds per session. That means that a sample does not represent a single session but a group of $N_{sessions}$ sessions. The figure 4.2 shows an example.

- traffic features variability: to partially solve this problem we need to capture a large amount of traffic to train the model, possibly collected in different times, different environments and under different conditions (e.g. take the call with different timings, alternatively turn on and off video and audio, even during the call, etc.). In addition, since the protocols keep being updated and the applications keep changing, the classifier must be kept up to date and possibly regenerated (by capturing again samples from the application that updated its protocol and doing again the training phase).

- different behaviours receiver-side and caller-side: in this case, if $N_{sessions}$ is the same in both sides the solution is easy, since it is enough to distinguish the two cases during training by capturing enough samples of both and defining two different classes. The problem is when even $N_{sessions}$ is different in the two cases since a single ML classifier cannot accept samples of different lengths.

- recognition of the application against the rest of the world: in addition to the samples representing the application we want to recognize we would need enough samples able characterize "the rest of the world", that in this case would at least be all the possible UDP traffic generated by other applications. Besides the complications in capturing such a large amount of traffic (also considering the fact that, since the training set needs to be balanced between classes, we would also eed as many samples from the application to recognize class), a sample is long $N_{sessions}$ since it represents the potential group of

$N_{sessions}$ sessions generated by the application under analysis. This translates into the fact that even the "world" class samples must be of the same length. So, the decision of articially constructing the "world" class samples was taken. It was first captured UDP traffic generated by other applications (like Jitsi [27], Skype/Zoom (if the application we want to detect is Zoom, Skype goes into the "rest of the world" and viceversa), other VoIP applications, etc.). Then, random combinations long $N_{sessions}$ of sessions taken from these captures are considered, translated into array of integers (where the integers are the usual number of packets in the first $N_{sec}$ seconds) and used as training samples for the "world" class.
In addition, even sessions generated from the application to detect were put into these combinations but only as long as one sample does not contain the features relative to all the $N_{sessions}$ sessions generated by the aforementioned application during the call/meeting (otherwise it would not go into the "world" class but into the application class). This is quite useful for the classifier since, because of how the macro-classifier works (in particular the combinator module), the ML classifier (which is the last stage) could receive samples where the features will only partially be relative to the application to detect while the others will be relative to other contemporary sessions and this sample must obviously be labeled as "world".

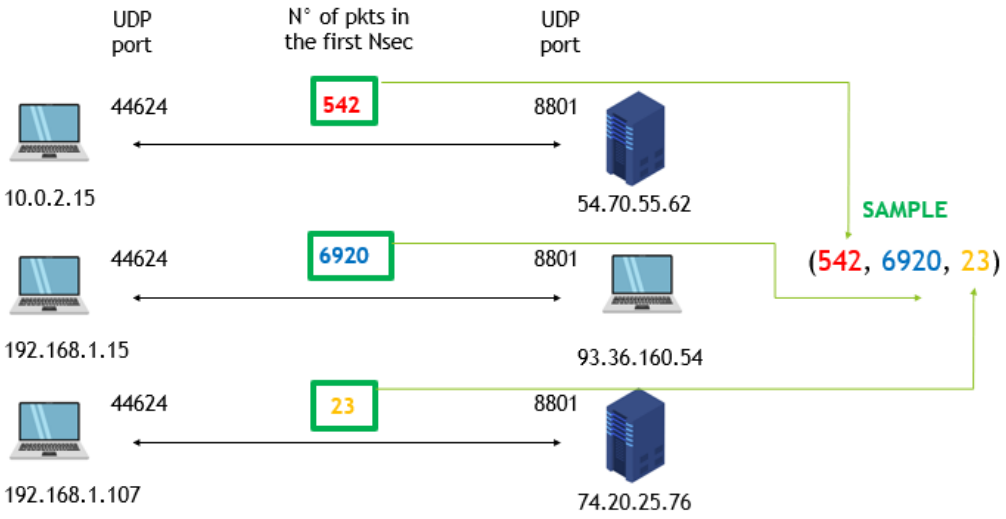


**Figure 4.2:** Example of construction of a sample from the UDP sessions ($N_{sessions}$=3)

To realize the ML classifier the scikit-learn [28] Python library was used.
Only two models were tested at the moment: Decision Tree and Naive Bayes. The main advantage of the first one is that is practically the only one that can be visualized, in particular as a directed acyclic graph (i.e. a tree) formed by nodes (boolean tests on the features) and branches (result of the test) and can thus show how the classification algorithm works.
The next subsections will show some examples.

### 4.4.1 Zoom

Since the number of packets per session is different when creating a meeting from when joining to one, it was taken the decision to differentiate the two cases (but just for the last stage of the macro-classifier, since we already saw that they share the same signature) by using for training, samples belonging to three different classes: zoom_join, zoom_create and world. This comes with another benefit: we are able not only to recognize if a Zoom meeting is being held but also discern if it is being created or joined to.

The figure 4.3 shows the Zoom Decision Tree trained with a total of a 150 samples, 50 samples per class (the classes being zoom_join, zoom_create and world) and with $N_{sec}$=15 seconds.

For each node in the tree we notice five (four if it is a leaf node) pieces of information:

- a test on one of the three features: to classify an unknown sample we must trace a path from the root node to one of leaf nodes where the path is chosen based on these tests. Note that the training input samples will be internally converted to float32 by the scikit-learn library function "fit" used to train the tree. That is why we see floating point numbers for the tests even if the features represent the number of packets for a session in the first $N_{sec}$ and that would be an integer. For example, in the first node we have "3° session <= 40.5" which means that if the third session of the sample to classify generated at most 40 packets in the first 15 seconds we have to visit the node to the left ("true" branch), otherwise we go to the right ("false" branch), and so on. We can notice how the test is missing from leaf nodes since when a sample arrives to a leaf has been already classified and must stop there.

- gini: it represents the value of the Gini of the current node (as explained in the appendix section A.1). It is a measure of the "purity" of the node. Obviously, the leaf nodes all have a Gini equal to 0 since they are the purest they can get (all samples in a leaf node belong to the same class).
  We can notice how the value of the Gini of the nodes decreases progressively as we go down the tree: this means that the far we are from the root node


**Figure 4.3:** Zoom Decision Tree, $N_{sec}$=15

the purer the nodes get (and that makes sense since during training we progressively apply new tests to divide the sets into smaller sets until they contain samples from only one class).

- samples: this is an integer which represents the number of samples (out of the entire number of samples used for training) that are contained in the current node. For example, in the root node we have 150 samples which is the entirety of the set. Following the "true" branch we can see how there are 91 samples which means that in the training set 91 samples out of 150 had their third session that generated less than 40 packets in the first 15 seconds.

- value: it is an array of integers where each integer represents how many samples of a certain class that were used for the training of the tree are contained in the current node. In order, we have the number of zoom_join, zoom_create and world samples which in the root node are all equal to 50 since it is better to have a balanced training set to avoid the model being biased towards a class over another.

- class: it is one of the three possible labels and represents the dominant class for the samples of the current node. For example, in the first node we do not have a dominant class since the set is balanced, thus it is indicated one of the three indiscriminately. Instead, if we look at the node reached following the "true" branch from the root node, the indicated class is zoom_create since we have 0 samples of zoom_join (which means that in the training set no sample relative to a join of a meeting had the third session that generated at most 40 packets in the first 15 seconds), 50 samples of zoom_create (which means that in the training set every sample relative to a creation of a meeting had the third session that generated at most 40 packets in the first 15 seconds) and only 41 samples of other type of traffic.

The fact that the Decision Tree can be visualized is useful since we can infer the characteristics of the three classes from the paths going from the root node to the leaf nodes.
Let's use the notation $np_{i-j}$ to indicate the number of packets for the i-th session in the first j seconds. Since a sample (which is basically an array of $N_{sessions}$ (3 for Zoom) integers) is sorted in descending order the first session is the one that generates the most number of packets in the first $N_{sec}$ seconds and so on.

Let us look at the leaf nodes: we can notice that almost for the entirety of the set of samples of zoom_join class (49 samples out of 50) $np_{1-15}$ is at most of 5247, $np_{2-15}$ is at most of 814 while $np_{3-15}$ is between and 41 and 47. Just one sample of the zoom_join class has a slightly different behaviour and thus follows a different path in the tree. Since it is a really small percentage it could be identified as "noise". For what concerns the zoom_create class, the entire set of samples used for training follows the same pattern: $np_{1-15}$ between 1610 and 3565, $np_{3-15}$ between 18 and 40 while the value $np_{2-15}$ is not indicated (probably because just the values of the other two features are sufficient to distinguish a creation of a Zoom meeting to the rest). As one could easily expect, the last class (world) is the most "noisy", since ideally it groups all the possible UDP traffic except for the one generated during a Zoom meeting and thus do not have a precise and fixed behaviour.

The table 4.1 sums this up (excluding the noisy samples because in really small number).

| Class | 1° session | 2° session | 3° session |
|---|---|---|---|
| zoom_join | $0 \leq np \leq 5247$ | $0 \leq np \leq 814$ | $41 \leq np \leq 47$ |
| zoom_create | $1610 \leq np \leq 3565$ | irrelevant | $18 \leq np \leq 40$ |

**Table 4.1:** Characteristics of Zoom traffic inferred by the Decision Tree ($N_{sec}$=15)

It is important to notice that these characteristics are more or less strict depending on the characteristics of the samples of the "world" class given for training. If, in fact, there were a lot of samples of generic UDP traffic really similar to the ones of Zoom traffic, the Decision Tree would result in a really deep tree with a lot of nodes and the conditions on the features would result much more stringent (i.e. narrower ranges for the number of packets per session).
The tables 4.2 and 4.3 show the same information but for different values of $N_{sec}$.

| Class | 1° session | 2° session | 3° session |
|---|---|---|---|
| zoom_join | $0 \leq np \leq 3428$ | $139 \leq np \leq 787$ | $0 \leq np \leq 47$ |
| zoom_create | irrelevant | irrelevant | $15 \leq np \leq 37$ |

**Table 4.2:** Characteristics of Zoom traffic inferred by the Decision Tree ($N_{sec}$=10)

| Class | 1° session | 2° session | 3° session |
|---|---|---|---|
| zoom_join | irrelevant | $81 \leq np \leq 742$ | $0 \leq np \leq 47$ |
| zoom_create | irrelevant | irrelevant | $17 \leq np \leq 26$ |

**Table 4.3:** Characteristics of Zoom traffic inferred by the Decision Tree ($N_{sec}$=5)

### 4.4.2 Skype

Not only, like Zoom, the number of packets per session is different when making a call from when receiving one, but also a different $N_{sessions}$ must be picked in the two cases. This results in the fact that two ML classifiers must be trained (since we

cannot have samples of different sizes for the same classifier) and each one needs samples from only two classes: Skype (alternatively receive and call) and world. For simplicity, let's focus just on the receiver side.
The figure 4.4 shows the Zoom Decision Tree trained with a total of a 120 samples, 60 samples per class (the classes being Skype and world) and with $N_{sec}$=15 seconds.

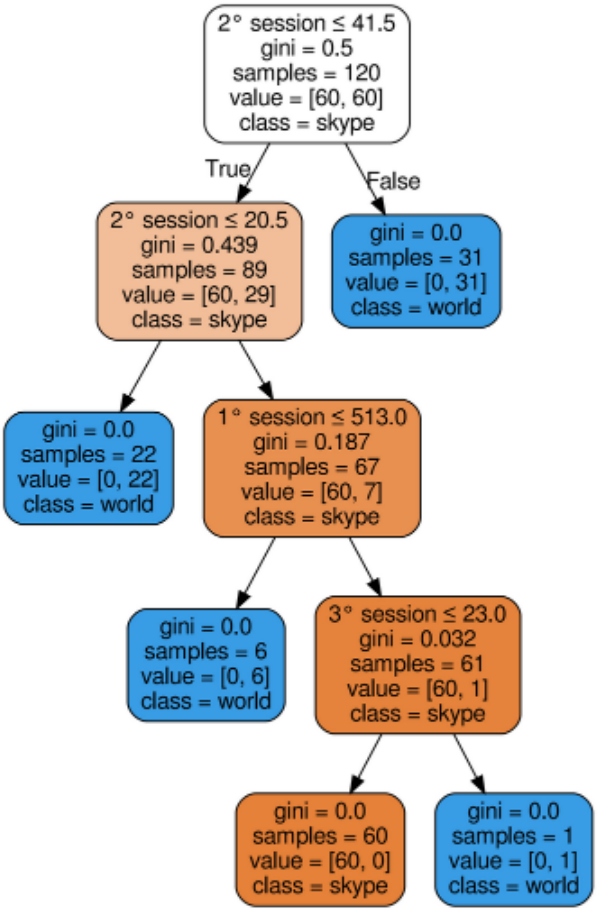


**Figure 4.4:** Skype (receiver) Decision Tree, $N_{sec}$=15

The first thing we notice is that this tree is less deep than the one related to Zoom. That is reasonable since we have samples from only two classes (and not three) and thus fewer tests are required. For the same reason, the Gini of the root node is 0.5 (same number of samples representatives of two classes).
We can see that there is only one leaf node labeled Skype (each one of the samples

of the Skype training set follows the same path from the root node to leaf).

The figures 4.4, 4.5 and 4.6 are some summary tables about Skype (receiver side) traffic characteristics inferred by the Decision Tree for different values of $N_{sec}$. As already mentioned in the Zoom subsection, this tests are to take with a grain of salt as they strongly depend from how many samples were used for training and from the quality of the world samples.

| Class | 1° session | 2° session | 3° session | 4° session |
|---|---|---|---|---|
| skype | np $\geq 514$ | $21 \leq np \leq 41$ | $0 \leq np \leq 23$ | irrelevant |

**Table 4.4:** Characteristics of Skype traffic (receiver) inferred by the Decision Tree ($N_{sec}$=15)

| Class | 1° session | 2° session | 3° session | 4° session |
|---|---|---|---|---|
| skype | np $\geq 475$ | irrelevant | $2 \leq np \leq 21$ | irrelevant |

**Table 4.5:** Characteristics of Skype traffic (receiver) inferred by the Decision Tree ($N_{sec}$=10)

| Class | 1° session | 2° session | 3° session | 4° session |
|---|---|---|---|---|
| skype | $183 \leq np \leq 1290$ | $29 \leq np \leq 36$ | np $\geq 2$ | irrelevant |

**Table 4.6:** Characteristics of Skype traffic (receiver) inferred by the Decision Tree ($N_{sec}$=5)

# Chapter 5

# Results

An important thing to take into consideration when validating the classifier is, obviously, to use samples which have not already been used for training. For this reason, usually a certain number of samples are collected and divided into two sets: one is the training set which is used to train the model; the resulting model is then tested with the samples coming from the other set of samples which is the testing set.
Clearly, the class of the samples of both sets must be known in advance since the trainer of the model needs both the samples and their labels, and since the testing phase consists in using the trained classifier to apply a label to each sample in the testing set and compare it with the real one. If the two labels correspond, that means that the classifier was able to find the correct class for the sample, otherwise we say that the sample got mislabeled and can thus fall into two different categories: false positives or false negatives.
In particular, a false negative is a sample generated by the application we want to detect (Zoom or Skype in this case) that, instead, does not get labeled as such by the classifier. A false positive is the exact opposite: a sample representing generic UDP traffic that gets labeled as Zoom or Skype.
The effect and the gravity of having false negatives and false positives strongly depends on the context in which the classifier happens to operate. If, for example, the field in which it operates is security-related (e.g. the classifier must find threats) false negatives are probably much more dangerous than false positives and can even prove to be fatal for the system. In fact, in such a case, a false positive would translate in a waste of time for an operator that would have to manually examine the sample or, in the worst case, in some kind of disservice since a benign flow could get blocked. Instead, a false negative is basically a threat that does not get detected and can thus operate undisturbed putting the entire system at risk.
Lastly, the overall accuracy of the classifier can then be computed as the number of testing samples that were correctly labeled by the classifier over the total number

of samples belonging to the testing set. This value is clearly useful because it represents the quality of the classifier but it is also important to consider what percentage of mislabeled samples are false negatives and what percentage are false positives because this gives us insight into the potentiality and the limits of the classifier.

Both models (Decision Tree and Naive Bayes) were tested, with different values of $N_{sec}$ and for both applications. Comparing the accuracies obtained with different models and different values for the parameters, besides being useful for discovering new properties of the traffic generated by the two applications, it also allows to find the best combination of model and parameters value to achieve the best value of accuracy. Clearly, one must then choose a tradeoff between accuracy and performance.

Firstly, the accuracy of the only Machine Learning classifier (which is the last stage of the macro-classifier) was tested. In particular, to test the Zoom ML classifier accuracy, a total of 200 samples were captured, 100 of them during a Zoom meeting while the other 100 being a mixture of generic UDP sessions like DNS, or traffic generated by other VoIP applications like Jitsi or Skype. The same 200 samples were used to test both models and for three different values of $N_{sec}$: 5, 10 and 15.
The table 5.1 shows the number of false negatives and false positives (expressed as a percentage with respect to the total number of samples in the testing set), followed by the overall accuracy.

| Model | $N_{sec}$ | False Positives | False Negatives | Accuracy |
|---|---|---|---|---|
| Decision Tree | 5 | 11% | 7% | 82% |
| Decision Tree | 10 | 9% | 5% | 86% |
| Decision Tree | 15 | 8% | 4% | 88% |
| Naive Bayes | 5 | 10% | 6% | 84% |
| Naive Bayes | 10 | 8% | 4% | 88% |
| Naive Bayes | 15 | 7% | 4% | 89% |

**Table 5.1:** Accuracy of different Machine Learning classifiers: Zoom

From the values in the table 5.1 some information can be inferred:

- we can see how the accuracy improves with the increase of seconds. That

means that, in general, exploiting a longer part of the flows helps the classifying process (at least until the first 15 seconds). But, another thing to notice is that this improvement in the accuracy is very light. In fact, we already have an accuracy of 82% for the Decision Tree and of 84% for Naive Bayes if we consider only the number of packets generated in first 5 seconds, and that improves of only 5-6% by increasing the number of seconds to 15.

- as it was already mentioned, the Decision Tree is not always one of the most accurate models. It is often used because it allows to visualize the generated model but it may not be too accurate, since it tends to overfit (and thus it does not generalize really well). In fact, the Naive Bayes model outperforms it by a few percentage points. Besides, even the Naive Bayes model works better if we consider the most seconds (i.e. 15) for the analysis of the number of generated packets.

- the number of false positives appears to be greater than the number of false negatives by quite a large amount. This is probably due to the fact that, while a Zoom sample has a certain pattern for what concerns the number of generated packets in the first $N_{sec}$ seconds and the variability of that pattern depends only on the state of audio and video and few other rare situations, a sample representing the "rest of the world" is much more variable because, ideally, it could be a sample generated by any application that generates UDP traffic. For this reason, it is more likely that one of the many other existing applications generate a certain number of packets that is somewhat similar to that of Zoom than having a Zoom meeting generate a number of packets that does not follow its usual pattern.

Finally, we can see how the best value of accuracy is achieved by using the Naive Bayes model with $N_{sec}$ equals to 15. However, the smartest solution is probably to choose to use the Naive Bayes model but with $N_{sec}$=10 since the improvement in the accuracy when increasing the number of seconds from 5 to 10 is of 4%, while when going from 10 to 15 seconds is just 1% and is thus not worth it. It is, in fact, possible to have a comparable accuracy and, at the same time, being able to achieve an earlier detection.

For the Skype (receiver side) classifier, a similar testing phase took place: 200 samples were captured, 100 when receiving a Skype call and 100 using other applications that generate UDP traffic (Jitsi, Zoom, etc.). Both models and with different values of $N_{sec}$ were tested.
An important thing to take into account is that, due to the fact that one session starts by immediately generating about six packets and then stops before resuming when the call is actually picked up, both the training of the model and the testing phase were conducted using samples relative to calls that were almost immediately

picked up. If the same classifier gets used to label a sample relative to a Skype call that was picked up after a few seconds, it probably won't label it as Skype since that particular session would have generated much less packets than usual.
That said, the table 5.2 shows the results of the tests done with the various Machine Learning classifiers for the detection of Skype samples.

| Model | $N_{sec}$ | False Positives | False Negatives | Accuracy |
|---|---|---|---|---|
| Decision Tree | 5 | 7% | 5% | 88% |
| Decision Tree | 10 | 7% | 5% | 88% |
| Decision Tree | 15 | 7% | 4% | 89% |
| Naive Bayes | 5 | 8% | 3% | 89% |
| Naive Bayes | 10 | 6% | 4% | 90% |
| Naive Bayes | 15 | 6% | 3% | 91% |

**Table 5.2:** Accuracy of different Machine Learning classifiers: Skype (receiver)

The Skype classifier appears to be more accurate than the Zoom one. That is probably due to two main factors:

- in the case of reception of a Skype call four sessions are generated, while during a Zoom meeting the number is three. Since a sample is an array long $N_{sessions}$ where the n-th element is the number of generated packets in the first few seconds by the n-th session, this results in the fact that in the case of Skype we have one extra feature. This extra feature probably helps in the recognition of the sample.

- in the case of Zoom, out of the three sessions only one is supposed to be a control session, while in the case of Skype the number of potential control sessions is three out four. During the analysis of the traffic it was found that these particular sessions already generate the most packets during the first 5 seconds but, more importantly, the number of packets they generate is much less variable than the number of packets generated by the sessions potentially transporting audio and video segment. This means that the corresponding features are more likely to stay in a certain range and thus make the Skype samples more distinctive.

For a similar reason, we notice how the accuracy only slightly improves with the increase of seconds, and this is even more visible than in the case of Zoom. In fact, during the analysis of the traffic it was found that the number of packets generated by three of the four sessions in the first 5 seconds is practically the same even if computed over the first 15 seconds. Only the last session keeps generating packets as the time goes on. This means that a potential handshake phase of the application gets (for the most part) already captured in the first 5 seconds and that is probably enough to achieve a good degree of accuracy.
Using for testing Skype samples relative to calls made under particular conditions (i.e. audio and/or video turned off, pick up later the call, etc.) makes the accuracy of the classifier drop dramatically. This problem can be tackled by diversifying even the training samples.
This time, the gap between false positives and false negatives is less pronounced, while Naive Bayes still outperforms the Decision Tree model. The best classifier is then the one that exploits the Naive Bayes model with $N_{sec}$=15 but, even this time, this probably isn't the smartest solution. If the cost of having many false positives is not so high we could choose the Naive Bayes model with $N_{sec}$=5 since the overall accuracy is already 89% and only 5 seconds are needed to identify the flow. Otherwise, $N_{sec}$=10 and a 90% could be enough (the number of false positives and false negatives are more balanced in this case).

After having tested the Machine Learning classifier alone, a series of tests were performed to test the accuracy of the whole macro-classifier, which is composed of both the signature detector and the ML classifier.
In particular, to estimate the number of false positives the two macro-classifiers (one for Skype and one for Zoom) were turned on to sniff and analyse the traffic for about a week when no Zoom meeting and no Skype calls were performed while other applications that generate UDP traffic were on. This way, it was sure that any possible detection was a false positive. For the measure of the false negatives, instead, the same 100 Zoom/Skype samples already used in the previous testing phase of the ML classifier were exploited.
The tables 5.3 and 5.4 show the number of false positives detected during the week of tests and the percentage of false negatives out of the 100 samples.

The first thing we notice is that the number of false positives dropped dramatically, since no false positive was detected, while the number of false negatives stayed the same.
This can be explained by the fact that, for false negatives (i.e. Skype and Zoom samples that were not detected), the addition of the signature detector does not change anything, since all the Zoom and Skype samples pass the signature detection test but can still happen for them to not be recognized by the Machine

| Model | $N_{sec}$ | False Positives | False Negatives |
|---|---|---|---|
| Decision Tree | 5 | 0 | 7% |
| Decision Tree | 10 | 0 | 5% |
| Decision Tree | 15 | 0 | 4% |
| Naive Bayes | 5 | 0 | 6% |
| Naive Bayes | 10 | 0 | 4% |
| Naive Bayes | 15 | 0 | 4% |

**Table 5.3:** Accuracy of the macro-classifier: Zoom

| Model | $N_{sec}$ | False Positives | False Negatives |
|---|---|---|---|
| Decision Tree | 5 | 0 | 5% |
| Decision Tree | 10 | 0 | 5% |
| Decision Tree | 15 | 0 | 4% |
| Naive Bayes | 5 | 0 | 3% |
| Naive Bayes | 10 | 0 | 4% |
| Naive Bayes | 15 | 0 | 3% |

**Table 5.4:** Accuracy of the macro-classifier: Skype (receiver)

Learning classifier, and since the same testing samples were used, the number of false negatives remained the same.
On the other hand, the false positives are practically reduced to zero because, this time, if one sample does not pass the signature detection phase first, it cannot proceed to the Machine Learning classifier and thus has not the possibility to be mislabeled by it. The probability for a generic sample to be labeled as Skype/Zoom is extremely lower since it has to both the same signature and generate a comparable number of packets in the first few seconds.
Basically, the addition of the signature detector is essentially useless for false

negatives since the ML classifier acts as a bottleneck (Skype/Zoom samples all pass the signature detector test but may still be mistakenly labeled by the ML classifier) while it is an extra "safety measure" for what concerns the false positives. In fact, for a sample to be considered "positive" (i.e. labeled as Skype/Zoom) that sample must both pass the signature detector test (and that is more unlikely if the considered signature is more detailed) and have the sessions generate a comparable number of packets in the first few seconds while for a sample to be considered "negative" it is enough that one of the two tests is not passed.

For what concerns the extremely low number of false positives, it must be taken into account that the macro-classifier was tested in a controlled environment, while in real-world situations it won't probably reach these levels of accuracy for different reasons:

- during the waiting phase when joining to a Zoom meeting, other three UDP sessions are generated and, unfortunately, they have the same signature of the other three sessions during the meeting and, sometimes, even a comparable number of generated packets in the first few seconds. For this reason, if the waiting phase lasts at least $N_{sec}$ seconds it could happen to have a double detection when, in reality, there is only one instance of a Zoom meeting.
- 200 testing samples probably aren't enough to properly test the accuracy: in particular, 100 sample cannot represent all the possible UDP traffic that could traverse the network. In fact, it could happen that another application (other than Skype and Zoom) that generates sessions with their same signature (even if really unlikely, especially if the signature is really detailed) exists, but simply was not included in the testing samples.
- the fact that the combinator stage of the macro-classifier selects all the possible combinations of $N_{sessions}$ sessions in a certain range of time could lead to some problems. For example, in the case of a Skype call, receiver side, we have the generation of four sessions, one between the two hosts, the other three going towards three servers (sometimes different, sometimes not) to the same UDP port 3478. For this reason, it would be enough to have another session (generated by other applications) going from the same host to a server on the port 3478 (and that is not rare, since it is the port used by the STUN protocol) that generates a number of packets comparable to one of the three client-server sessions of the Skype call to have a multiple detection and thus a false positive.

The number of false positives can also be influenced by possible contemporary Zoom meetings or Skype calls.
Let's take as an example the creation of a Zoom meeting. As it was already widely

discussed, three UDP sessions having a fixed signature are generated during the creation of a Zoom meeting, and these three sessions always generate a comparable number of packets in the first few seconds.
That means that three generic UDP sessions are considered to be generated by an instance of a Zoom meeting if, and only if, they are not too many seconds apart (i.e. 1 second max), the IP addresses and ports used by the sessions reflect the application signature and the corresponding sample (i.e. the array containing the number of packets generated by each sessions in the first few seconds) gets labeled as being "Zoom" by the Machine Learning classifier.
Now, let's suppose we have the contemporary creation of two different Zoom meetings. In this case, a total of six sessions would be generated and, under some unfortunate conditions, more than two detections (which are the real ones) could happen. In particular, the following events must all be happening at the same time:

- they must be near in time, in the worst case all the six sessions in a range of 1 second at most.
- the IP address of the server used for the two meetings must be the same (while for the server port we are sure that this always happens, since it is fixed to 8801)
- the hosts creating the meeting must be behind a NAT router and the tool capturing and analysing the traffic must not be inside the same network of the two hosts: this way, the tool would have no way of distinguishing the two hosts (the used port is of no use since it changes from one sessions to another even for the same meeting)

The coexistence of these three conditions could be a problem because the function of the combinator stage of the macro-classifier is to form groups of $N_{sessions}$ (which is three for Zoom) sessions from a larger group of sessions that are near in time. Since the order is not important and we do not want repetitions, these "groups" of sessions are the k-combinations without repetitions of a larger set S containing n elements, and the number of these combinations is equal to the binomial coefficient:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Now, if the first condition is valid, our set S would be the set containing the sessions generated by both meetings, n would be six while k would be three; thus, the number of combinations of three sessions formed by the combinator stage would be:

$$\binom{6}{3} = \frac{6!}{3!(6-3)!} = 20$$

Then, if even the other two conditions are valid, out of the 20 possible combinations the chain of signature detector and ML classifier would mistakenly detect more than two Zoom meetings.
In fact, the test of the signature detector would be passed by each one of the 20 combinations (since they all have the same signature) while the ML classifier helps in excluding part of these 20 combinations but not all of them. In particular, if the three sessions generated during the creation of the meeting would be called session A, B and C, the samples that will get labeled as Zoom are the ones relative to the combinations that contain a session A, a session B and a session C even if generated by two different meetings, since the number of generated packets in the first few seconds is the only information we are interested in and that does not change too much from meeting to meeting.
In this case, the number of detected Zoom meetings would be eight, where two are the real ones while the other six correspond to combinations that contain the right sequence of sessions but generated by two different meetings.
Obviously, the same applies for Skype, but this time the server used in the two calls can also be different since more than one server is contacted even during the same call. Clearly, the number of total combinations and the number of the ones that get mistakenly detected is different from the Zoom case. Besides, the situation gets even more complicated if more than two contemporary Zoom meetings or Skype calls are being held.

Finally, from these tables, another important finding was made, which is the so called "paradox" of the macro-classifier. In fact, the accuracy of the signature detector alone basically gets not affected or gets even worse because of the addition of the ML classifier stage.
In particular:

- false negatives: having false negatives when using the macro-classifier without the ML classifier stage is practically impossible since, this time, for a sample to be labeled as Skype or Zoom it is enough for it to have the same signature in terms of number of generated sessions and characteristics of IP addresses and UDP ports. Since these features (or at least a subset of them) were proven to be fixed, Skype and Zoom samples tend always to be detected.
  The only cases in which we could have some false negatives are if the protocol used by the applications gets changed and with that changes even the generated traffic or if the signature of the application to detect is not fixed but can vary. The latter is not necessarily a problem, since that's even the case of Skype and the chosen solution was to simply select a subset of these features that remain fixed; in the event that the entire signature varies, the signature detector would simply be useless and another strategy should be used.
  Adding the ML classifier at the end of the chain of the macro-classifier would result in an increase in the number of false negatives since this time a

Zoom/Skype sample must now pass even the ML classifier test to be correctly detected (and, unlike the signature, the number of generated packets per session may vary a bit, especially because of different conditions under which the meeting/call can be held).

- false positives: the number of false positives found by the macro-classifier with only the signature detector basically does not get affected by the addition of ML classifier, since it is practically zero in both cases. In fact, at least in our controlled environment, a sample that was not Zoom or Skype already had a different signature so that was enough to label it as being generic UDP traffic. The addition of the ML classification is useless because most of the generic UDP samples basically never reach it.

It should be noted, however, that in real world situations, our macro-classifier without the ML classifier phase will probably find some false positives. For example, in the usual waiting phase during a join to a Zoom meeting.
Besides, when dealing with contemporary meetings and calls (but only if the hosts are behind a NAT router) the situation would be worse since, for example, in the case of two contemporary Zoom meetings, all of the 20 combinations would be detected as being 20 different instances of a meeting (instead of the 8 detections made by the complete macro-classifier). And to make matters worse, since the number of generated packets would be no longer considered, it would be enough to have even only one contemporary session (generated by other applications) that share the same signature with one of the session generated by the application to detect, to have multiple fake detections (while with a complete macro-classifier, this extra session should also have a number of generated packets in the first few seconds comparable to the session of the application to detect, and that would be more unlikely).
To conclude, it must be said that, even if the addition of the ML classifier to the chain of the macro-classifier seems to be useless or even detrimental, in some cases it could prove to be useful. In fact, even if not so common, some other applications could share the same signature with Skype and Zoom but it is extremely unlikely to share even the number of generated packets per session. Or, more likely, a malicious application could purposedly act as Zoom or Skype (in terms of generated sessions, IPs and ports) but the number of generated packets would probably be different since the exchanged data are not the same.
The use of the ML classifier along with the signature detector should then be carefully assessed by the final user given its limits and potential.

# Chapter 6

# Conclusion

This work attempted to solve the problem of recognition of a certain VoIP application (in particular, Skype and Zoom) through the analysis of the generated traffic. Specifically, the goal was set to detect a Skype call or a Zoom meeting by exploiting only the traffic generated in the first few seconds. The fact that only the early generated traffic must be considered is driven by the fact that a tool of this type can, for example, be deployed in a corporate setting to perform real-time blocking, filtering or labelling of traffic flows for QoS enforcement.
The traffic classification issue attracted a lot of attention from the experts in the field and that is testified by the fact that many solutions have been proposed and accurately tested. Some of them then proved to be unfeasible (e.g. port-based classification and deep packet inspection, mainly because of the use of dynamic ports and ciphered payloads) while others proved to be useless for our case, which is early recognition (e.g. exploitation of flow-level information, because it would need the entire flow to finish).
Before designing the tool, the UDP traffic generated by the two applications in question was analysed to discover the number of generated UDP sessions, the number of different IP addresses and of different UDP ports used; in addition, other features were computed under different conditions to determine the nature of the different sessions: in particular, the number of generated packets per session in the first $N_{sec}$ seconds and the variation of the throughput per session were computed. The information obtained from these tests allowed to make some assumptions about the nature of the sessions and were then used for the design of the so called macro-classifier, composed of two main phases: the signature detection phase and the classification phase.
In the first one, the information about sessions, IPs and ports (which is the "signature") is exploited. For example, the signature of the traffic generated during the creation of a Zoom meeting is three sessions that use only two endpoints (i.e. the host creating the meeting and a server), with the user port constantly changing

per session and the server port being fixed to 8801. If a group of three sessions has these characteristics passes the test of the signature detector and can thus proceed to the classification phase. In this second phase, a previously learned Machine Learning model is used. The chosen features are the number of packets per session in the first $N_{sec}$ seconds and both a Decision Tree and a Naive Bayes model were tested, with different values of $N_{sec}$.
The main findings are the fact the Naive Bayes model is the one that performs better, and that usually a greater $N_{sec}$ is better (at least until a value of 15 seconds). The accuracy of only the ML classifier (Naive Bayes, $N_{sec}$=15) obtained testing a total of 200 samples was of 89% for Zoom and of 91% for Skype (receiver side). The addition of the signature detector helps a lot in the recognition of false positives since it is rare for two different applications to share the same fixed signature (while the number of generated packets is more variable and thus ambiguous). Some tests performed with only the signature detector revelead the so called "paradox of the macro classifier": adding the ML classifier after the signature detector does not affect its accuracy or even makes it worse. In fact, the false positives basically remain the same (samples not being Zoom/Skype get already blocked by the signature detector and the ML classifier is useless) while the false negatives even increase (it is not enough for a Zoom/Skype sample to have the prefixed signature, but it also has to pass the test of ML classifier). For this reason, the use of the ML classifier in conjunction with the signature detector should be done with care and only under particular conditions.

The developed tool still has some open issues that might need to be solved:

- in the case of a Skype call there is one problem with a particular session: the session 4 in the receiver side (or the corresponding session 6 in the caller side) which is the session that potentially transports the audio and video segments since it generates a large number of bytes. The problem with this session is that it starts generating packets immediately when the call starts ringing by generating about 6 STUN packets and then stops until the call is picked up. Since the $N_{sec}$ seconds countdown timer starts when the first packet of the session arrives, the amount of packets generated during the actual call that are captured during the first $N_{sec}$ seconds depends from when the other user responds to the call. In fact, in the worst case, if the other user does not respond within $N_{sec}$ seconds the number of generated packets by that session in the first $N_{sec}$ seconds would be only six.

- it was already pointed out how, for Skype, the choice of a good value for $N_{sessions}$ is not so easy if we want to recognize both the receiver and the caller side of the call. So, in such a case, at least for the moment, the only solution would be to use two different classifiers for the two cases since a classifier

does not accept samples with different lengths (and the length of the samples corresponds to the chosen value of $N_{sessions}$).

- in the case of join to a Zoom meeting we have three extra sessions and the number of generated packets per session in the first $N_{sec}$ seconds of these three extra sessions could sometimes be comparable to the same info relative to the three sessions generated during the actual meeting.
  Since the first three sessions also the same signature, if the host of the meeting does not accept the user trying to join in less than $N_{sec}$ that could result in a double detection of Zoom despite having only one ongoing instance of a meeting.

- because of the combinator phase that builds all the possible combinations of $N_{sessions}$ sessions, the classifier could not scale too well both in the case when too much traffic is being generated and in a future possible case when we would consider an application that generates many sessions.
  Fortunately, the combinations are built if and only if they have the required signature (otherwise we stop there).

- the Internet with all its protocols and network applications constantly change and get patches and updates. This could potentially be a problem since if the made changes are significant, the whole macro-classifier (or at least the Machine Learning classifier) must be redesigned and tested again. The good news is that, even if updates comes very often, they don't usually change very much in the behaviour of the applications so no changes in the classifier are required. In general, samples generated by the applications under analysis must be periodically captured and studied to detect possible changes in the protocol.

- the tool struggles when dealing with contemporary meetings/calls if the hosts making these calls are behind a NAT router and the the tool is not in the same network. In fact, in this case it would be impossible for the tool to distinguish the different hosts and because of the combinator phase more than the real number of actual ongoing calls would be detected.

Besides, the work can still be improved and enhanced to increase the accuracy or to add new functions:

- each Machine Learning model has its own pros and cons and, for that reason, is suited only to a particular type of problems. This means that we cannot use whatever model we like anytime a problem needs to be solved since the resulting accuracy could be lower than we expect.
  For this reason, it is always a good practice to follow this strategy: first, the

problem must be accurately studied to understand what we want the model to do, what is the the cost of possible false positives and false negatives (i.e. is it really bad if a sample which does not represent the application under analysis is detected as such?), what are the priorities, if performance (a quick recognition or a quick learning phase) or accuracy and so on.
Then, the features that can accurately describe a sample must be chosen. And last, but not least, a Machine Learning model must be picked. One can first choose a few of them, based on the characteristics of the problem to solve and then test each one of them before finally choosing to deploy the one performing better than the rest.

- the choice of the features is probably one of the most important phase when working with Machine Learning algorithms. In fact, even the quality of the model depends on the features that were picked. In general, a good strategy might be to choose a set of features that are statistically independent from each other but strongly dependent from the application behaviour. To try to improve the developed classifier other features can be added or even replace those currently in use.

- as it was already mentioned, only the UDP packets were considered during the analysis of the traffic generated by Skype and Zoom, and thus even for the design of the macro-classifier. A possible improvement in the macro-classifier could be to include the TCP packets into the analysis looking for possible regular patterns or particular characteristics.

- since the accuracy of the classifier drops if the testing samples are captured under different conditions from those under which the training samples were captured, a good strategy is to diversify the training set: first of all, the number of training samples can be increased, and the new samples should be captured under different conditions (e.g. pick up the call after a few seconds, turn off the video and/or the audio, etc.) to make the classifier more resilient to a possible variability in the features.

- finally, the macro-classifier can be extended to detect applications other than Zoom or Skype.

# Appendix A

# Machine Learning fundamentals

The power of machine learning algorithms is the ability to generalize from examples and thus to generate programs from data instead of having to manually construct them.
In general, given a set of n samples of data as input (which is called the training set) a learning problem consists in trying to predict certain properties of unknown data.
Each sample can be represented as a vector, where each element is called a feature (or an attribute).
As we can see in [29], machine learning algorithms can be divided into two categories:

- supervised learning: each sample comes with the desired output and the algorithm infers a function that maps a new unknown sample to the correct output. If the output can only be one of a finite number of discrete categories we talk about a classification problem; it is a regression problem if the desired output consists of one or more continuous variables

- unsupervised learning: the training data is not pre-labeled. In this case, the goal is usually to find groups of similar examples within the data (clustering)

Here follows a brief description of the fundamentals of the two models used in this work.

## A.1 Decision Tree

Given training vectors and label vectors, a decision tree learner recursively partitions the space such that the samples with the same labels are grouped together.

In particular, starting from only one node that contains all the samples of the training set, for each node and for each candidate split (which consists in the choice of a feature F and a threshold T) the data are partitioned in two sets (so that F <= T for one set, and F > T for the other).
The split that minimises an impurity function (which depends on the task being solved) is chosen. The learning process continues until a maximum depth is reached, we have a minimum number of samples or all samples belong to the same class.
The process of labeling an unknown sample consists in following a path from the root node to one of the leaf nodes, since each leaf represents a class.
The Gini of a node measures the probability of a sample being wrongly classified

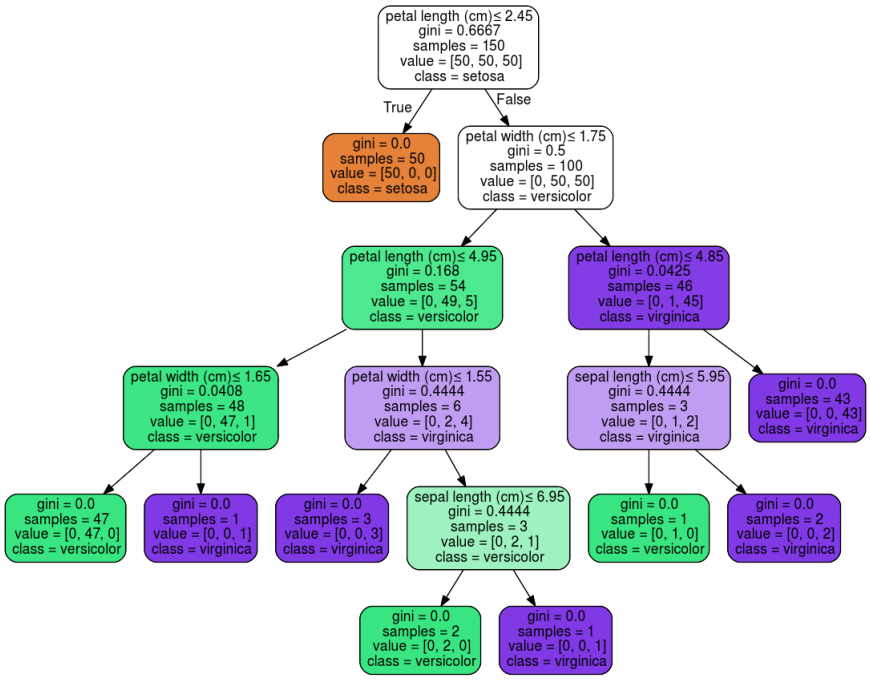


**Figure A.1:** Example of a Decision Tree generated with scikit-learn

when it is randomly chosen amongst all the samples belonging to the node itself. If N is the number of classes and p(i) is the probability of picking a sample with class c, the Gini of the node is computed as:

$$G = 1 - \sum_{i=1}^{N} (p(i))^2$$

It can assume all the possible values between 0 and 1: if all the elements in a node belong to a single class then the Gini is 0 and the node can be defined pure, while if the elements are randomly distributed across various classes the Gini is 1.

As stated in [30], some of the advantages of decision trees are:

- since trees can be visualized ("white box" model), results are simple to understand and to interpret
- requires little to no data preparation, while other techniques often require at least data normalization
- the cost of predicting data is logarithmic in the number of data points used to train the tree

On the contrary, some of the disadvantages of decision trees are:

- they are subject to a problem called "overfitting", which is when over-complex trees that do not generalize the data well are generated. This problem can be mitigated by "pruning" the tree (i.e. removing sections of the tree that help less in classifying samples) or by setting a maximum depth for the tree
- even the smallest variation in the training data can lead to a completely different tree. Using decision trees within an ensemble can be a possible solution
- since decision tree learners create biased trees if classes are not evenly distributed, it is recommended to balance the training set before using it to fit the decision tree
- since the problem of learning an optimal decision tree is known to be NP-complete, in practice decision tree learners exploit simpler heuristic algorithms which cannot guarantee to return the globally optimal decision tree

## A.2 Naïve Bayes

As clearly and briefly explained in [9], Naïve-Bayes is based on the Bayes' theorem:

$$P(A|B) = \frac{P(A) * P(B|A)}{P(B)}$$

where A and B are two events, P(A) and P(B) are the probabilities of the events A and B occurring while P(B|A) is the conditional probability of A occurring given that B is true.
Now, given a vector X of instances where each instance is described by a set of features (and relative values) and by a random variable C representing the class, this classification technique analyses for each instance the relationship between

each feature and the class to derive a conditional probability.
In particular, the probability that an instance x belongs to a class c is:

$$P(C = c|X = x) = \frac{P(C = c) * P(X = x|C = c)}{P(X = x)} \tag{1}$$

where P(C=c) is the probability of a certain class and it is computed during training by counting how many times it occurs in the training set, while, under the assumption that the attributes are independent (that is why this technique is called "Naïve"), the following is valid:

$$P(X = x|C = c) = \prod_i P(X_i = x_i|C = c)$$

which is the product of the probabilities of each single feature.
It is important to notice that the equation (1) is only applicable if the features are qualitative (i.e. they take a small number of values): in this case, the probabilities can then be estimated from the frequencies of the instances in the training set, while in the case of quantitative features, discretisation may be used.
Lastly, P(X=x) is an invariant across classes and therefore only necessary as a normalising constant. It can be computed as:

$$P(X = x) = \sum_j P(C_j)P(X = x|C_j)$$

In practice, when dealing with real Naïve-Bayes classifiers (like the ones realized with scikit-learn [31]) a certain sample is labeled with the class $\hat{c}$ such that:

$$\hat{c} = \arg\max_c P(c) * \prod_i P(x_i|c)$$

Different Naïve-Bayes classifiers exist since each one makes different assumptions from the others about the distribution of $P(x_i|c)$.
One of them (which is the one used in this work) is the Gaussian Naïve-Bayes classifier, where the probability of the features is assumed to be Gaussian:

$$P(x_i|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} * exp(-\frac{(x_y - \mu_c)^2}{2\sigma_c^2})$$

The parameters $\sigma_c$ and $\mu_c$ are estimated using maximum likelihood.
Despite the naive assumptions it is based on, this type of classifier has been proven to work quite well in various situations, like spam filtering and document classification.
Some of the advantages of this type of classifier are:

- only a small amount of training data is needed
- it is faster than most of the other classifiers
- since each class-conditional feature distribution can be independently computed as a one dimensional distribution, this helps to solve problems related to the curse of dimensionality

However, despite being a modest classifier, it is instead a bad estimator.

# Bibliography


[1] Joe Hallock. «A Brief History of VoIP, Document One – The Past». In: Nov. 2004 (cit. on p. 1).

[2] *VoIP Wikipedia page*. URL: `https://en.wikipedia.org/wiki/Voice_over_IP` (cit. on p. 1).

[3] Deanna Tanner Okun, Charlotte R. Lane, Daniel R. Pearson, Shara L. Aranoff, Irving A. Williamson, and Dean A. Pinkert. «Recent Trends in U.S. Services Trade, 2010 Annual Report». In: (June 2010) (cit. on p. 1).

[4] *Skype Website*. URL: `https://www.skype.com` (cit. on p. 2).

[5] *Zoom Website*. URL: `https://zoom.us/` (cit. on p. 2).

[6] Ray Tiernan. *Is There Room for Zoom Video to Continue Zooming Upward?* URL: `https://www.thestreet.com/investing/can-zoom-continue-zooming-upward` (cit. on p. 2).

[7] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. «Traffic classification through simple statistical fingerprinting». In: *ACM SIGCOMM Computer Communication Review* (Jan. 2007) (cit. on p. 6).

[8] Tom Auld, Andrew W. Moore, and Stephen F. Gull. «Bayesian Neural Networks for Internet Traffic Classification». In: *IEEE Transactions on Neural Networks* (Jan. 2007) (cit. on p. 6).

[9] Nigel Williams, Sebastian Zander, and Grenville Armitage. «A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification». In: *14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)* (Sept. 2010) (cit. on pp. 6, 66).

[10] H. Dahmouni, S. Vaton, and D. Rossé. «A markovian signature-based approach to ip traffic classification». In: *Proceedings of the 3rd annual ACM workshop on Mining network data*. June 2007, pp. 29–34 (cit. on p. 6).

[11] C. Wright, F. Monrose, and G. M. Massoń. «Hmm profiles for network traffic classification». In: *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*. Oct. 2004, pp. 9–15 (cit. on p. 6).

[12] *Internet Assigned Numbers Authority (IANA) registry*. URL: `http://www.iana.org/assignments/port-numbers` (cit. on p. 6).

[13] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. «A Survey of Payload-Based Traffic Classification Approaches». In: *IEEE Communications Surveys  Tutorials* 16 (2014) (cit. on p. 7).

[14] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wanǵ. «Accurate, scalable in-network identification of p2p traffic using application signatures». In: *Proceedings of the 13th international conference on World Wide Web*. May 2004, pp. 512–521 (cit. on p. 7).

[15] *L7filter*. URL: `http://l7-filter.sourceforge.net/` (cit. on p. 7).

[16] Kei Takeshita, Takeshi Kurosawa, Masayuki Tsujino, Motoi Iwashita, Masatsugu Ichino, and Naohisa Komatsu. «Evaluation of http video classification method using flow group information». In: *2010 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)* (Sept. 2010) (cit. on p. 8).

[17] Blake Anderson and David McGrew. «Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity». In: *KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Aug. 2017 (cit. on p. 8).

[18] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegeĺ. «BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection». In: *CoNEXT '12: Proceedings of the 8th international conference on Emerging networking experiments and technologies*. Dec. 2012, pp. 349–360 (cit. on p. 8).

[19] Robin Sommer and Vern Paxson. «Outside the Closed World: On Using Machine Learning For Network Intrusion Detection». In: *2010 IEEE Symposium on Security and Privacy* (May 2010) (cit. on p. 8).

[20] Y. Chen and K. Hwang. «Spectral analysis of TCP flows for defense against reduction-of-quality attacks». In: *2007 IEEE International Conference on Communications* (June 2007) (cit. on p. 8).

[21] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. «Traffic classification on the fly». In: *ACM SIGCOMM Computer Communication Review* (Apr. 2006) (cit. on pp. 8, 9, 12).

[22] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. «Revealing Skype Traffic: When Randomness Plays with You». In: *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. Aug. 2007, pp. 37–48 (cit. on p. 10).

[23] *Wireshark website*. URL: `https://www.wireshark.org/` (cit. on p. 13).

[24] *tcpdump manpage*. URL: `https://www.tcpdump.org/manpages/tcpdump.1.html` (cit. on p. 13).

[25] *Session Traversal Utilities for NAT (RF 5389)*. URL: `https://tools.ietf.org/html/rfc5389` (cit. on p. 21).

[26] *Joblib website*. URL: `https://joblib.readthedocs.io/en/latest/` (cit. on p. 34).

[27] *Jitsi website*. URL: `https://meet.jit.si/` (cit. on p. 40).

[28] *Scikit-learn website*. URL: `https://scikit-learn.org/stable/` (cit. on p. 41).

[29] *An introduction to machine learning with scikit-learn*. URL: `https://scikit-learn.org/stable/tutorial/basic/tutorial.html` (cit. on p. 64).

[30] *Scikit-Learn Decision Trees*. URL: `https://scikit-learn.org/stable/modules/tree.html` (cit. on p. 66).

[31] *Scikit-Learn Naïve-Bayes*. URL: `https://scikit-learn.org/stable/modules/naive_bayes.html` (cit. on p. 67).