UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

FINAL DISSERTATION

# Statistical Flow Classification for the IoT

Supervisor

Prof. Roberto Passerone

Student

Gennaro Cirillo

Academic year 2018/2019

# Contents

# Abstract

Internet Of Things in these years is becoming an increasingly studied topic. We define Internet Of Things as a variety of technologies that allow us to connect objects to the network. For instance, we can connect cameras, sensors, speakers and use them over long distances. In the next years connect objects to the network will be widespread, and the connections will grow up every year. Just think to smart cities, autonomous driving and robotics, in the future they will impact more and more in our lives. Internet traffic is constantly increasing, the devices connected to the network grow up very fast every year. With the introduction of 5G networks, the connections performace will be increased, allowing us to use more devices together, and reduce the latency time. You can image to have multiple connection at the same time, for example, you can use your smartphone to control the temperature in your house, check your security camera, control in real time the situation of the traffic on the road and many other things. Sensors connected to the network, lately are used for the security and maintenance of our roads, monitoring the stability of bridges and viaducts. IoT devices can be employable in a lot of different fields, from medical and public security to simply for all the things we do everyday, improving our quality of life. Therefore, the Internet of Things (IoT) holds a great potential for the development of innovative applications. Hence the need to filter and dispach the traffic to optimise our networks, we want fast and reliable connections, therefore we need methods to separate different types of communications. These devices uses specific protocols, called "IoT procotols", they have different features compared to the traditional protocols. For this reason, network operators must support these devices with differentiated services, which rely on the ability to automatically recognize and classify the nature of the communication flows. Mobile network operators have implemented systems that allow to separate different traffic from different services. For instance, you can count only the amount of data used by a set of specific applications (e.g. Facebook, Whatsapp, Instagram). Furthermore, methods to control the internet flows are used to check protocol anomalies, propagations of malware, or simply collect statistical data on the use of the networks themselves. Analyzing the flows instead of the single packets allow us to scan encrypted communications. This kind of analysis is called "deep flow inspection", the classification is made by considering the overall behavior of the flow of packets. In our case we consider different types of features to compute the classification (e.g. packet length fraction, round trip time, frequency of sent and received packets). In this thesis we discuss statistical methods to analyze packets flows and classify them as traffic that belongs to IoT devices or to traditional, non-IoT communication. The methods work by operating on a training dataset. We start by presenting the methods employed to collect the flow data used in the study, which comes from both simulation and from real deployments. Then we give details regarding the tools used for capturing, analyzing and classifying the flows. We use two types of flows attributes for the training datasets, the first set of attributes comes from the analysis of the packets statistics (e.g. acknowledgment packets fraction, packets length fraction), the second set of attributes comes from the analysis of the packets frequency. Then we present an overview of the machine learning algorithms used in our analysis. After that we will see the various datasets employed coming from different sources. We then present our results. For the first set of attributes regarding the packets statistics, we mainly employ two methods for the classification: a clustering approach, which learns directly from the structure of the dataset; and a classification tree, trained with the collected data and evaluated using 10-fold cross validation. The results show that classification tree outperform clustering on all datasets, and achieve high accuracy on both homogeneous simulated and real deployment traffic data. For the second set of attributes regarding the packets frequency, we show different techniques used to collect the temporal series and show that most of them doesn't performs well for the classification. We will look also into different preliminary experiments and their results. Finally we show with more focus the method that achieve the best result. We construct the time series using the packets length. For the classification we use an ensemble learning algorithm that uses a Random Forest classifier. We train the trees with the entire

frequency spectrum, given by the outcome of the Fast Fourier transform. The evaluation is performed using both 10-fold cross validation and a split between training, validation and test-set. The latter is used for hyperparameter tuning. The results show that for reasonably large datasets the classifier achieves very high accuracy, as well as Precision and Recall rates.

# 1 Introduction

Progress in wireless network connectivity, miniaturization, and computing resources with advanced learning capabilities have seen the proliferation in the last decade of ubiquitous, independent and mostly autonomous devices dedicated to new and diverse applications, including sensing, remote control, fleet tracking, and environmental monitoring and conditioning. Network operators and component manufacturers must support this array of applications by providing network management functions such as resource planning, quality of service (QoS) provisioning, load balancing and lawful intrusion detection. At the same time, the diversity of IoT devices and applications results in infrastructure requirements that are vastly heterogeneous, posing significant challenges in delivering an optimized set of differentiated services. IoT traffic could be very diverse in terms of network parameters and bandwidth requirements, reaching the very extremes of the available range. On one side, some devices may be monitoring quantities that change infrequently, such as a power meter or a temperature sensor. These devices would therefore connect to the network from time to time, and exchange minimal amount of data. The access pattern may be periodic, such as the power meter, or sporadic, triggered by the data itself, when the system is interested in detecting certain events. Latency in these cases is often of little concern, as long as it is reasonable. However, more and more often, a guaranteed connection may be required in order to quickly trigger an alarm, such as the detection of hazardous events. On the other side of the spectrum, data may be streamed in large amounts, for instance by surveillance cameras or by telemetry systems on vehicles. One additional aspect is the rapidly increasing and large number of devices which may require the connection. Considering that communication between things may be less tolerant to latency, errors and failures than a corresponding human communication, and together with the needed service guarantees, which are not currently sufficiently supported by the network providers, service providers could be prompted to shift their communication products which need to evolve from a simple, undifferentiated dumb pipe to a more dedicated, performance aware network [30]. Traffic classification is therefore an essential feature to retrofit existing networks with devices that can support extra and smart functionalities. In addition, the value created by things is not limited to their specific functionality, but is more concentrated on the information that they produce. This information often acquires meaning when aggregated with the data produced by a multitude of devices. For these reasons, information processing does not generally occur on the individual devices, but is rather delegated to geographically distributed servers, known as *cloud services*, whether or not the device is technically able to handle the data. Communication is therefore not just limited to data which gets collected in a central repository, but is instead a key part of the functionality of a large integrated and distributed system. A network that is aware of the services can therefore take decisions that can help optimize the overall performance.

Detecting and classifying traffic can be accomplished using several techniques. *Deep packet inspection (DPI)* consists in observing the contents of the individual packets in order to detect characteristic patterns that identify the kind of protocol in use [24, 54]. The simplest methods work at the network level (TCP or UDP) and consider IP addresses and port numbers, which in most cases are individually assigned to particular services. More complete methods analyze also the higher levels of the protocol, up to the application layer, to avoid problems with port spoofing and to provide a more general approach. From the protocol, one can then deduce the class of service that is being used, and therefore classify the underlying application. An alternative approach is *deep flow inspection (DFI)*, which considers the overall behavior of the flow of packets, by analyzing, for instance, the size of the packets, their inter-arrival times, and other statistics. Deep flow inspection typically uses statistical classification and machine learning to provide a result. Because of the working principle of DFI, this method is also known as *behavioral classification*. Behavioral classification is based on features of flows, rather than of individual packets, and uses statistical or behavioral classifiers to determine the application or protocol in use. Statistical and behavioral classifiers work at slightly different levels

of abstraction, the first looking at features of the packets, the latter focusing more on the flow as a whole, looking at the access patterns. Their advantages are similar. The benefits of statistical and behavioral classification are twofold. First, behavioral classification can be applied whenever packets cannot be inspected, either because of the use of encryption or for privacy restrictions. In the second place, behavioral classification may provide results also when applications use standard application level protocols, such as HTTP, to exchange information. Valenti et al [46] provide a comprehensive review of statistical and behavioral methods, discuss the operation of support vector machines and decision trees, and analyze in detail both Kiss, a statistical classifier [10], and the Abacus behavioral classification algorithm [37]. In particular, the statistical classifier looks at the entropy of groups of four bits in the headers, building a specific signature for each application. Abacus, on the other hand, looks at the connection activity of the application, considering number of connections, frequency of connections and stability of the link. For behavioral classification to work efficiently, one has to identify the most specific characteristics of the flows to be identified, and then employ a correct classification methods.

## 1.1 Characteristics of M2M traffic

The first aspect that must be considered when dealing with behavioral classification is an analysis of the communication pattern of the traffic that we want to classify. Shafiq et al. [33] have conducted a set of measurements to compare the machine-to-machine (M2M) traffic to traditional cellular smartphone traffic. The dataset comprises flows exchanged over the cellular data network in the USA. They report, for instance, that M2M devices have a much larger ratio of uplink to downlink traffic volume, their traffic typically exhibits different diurnal patterns, they are more likely to generate synchronized traffic resulting in bursty aggregate traffic volumes, and are less mobile compared to smartphones. One interesting contribution of this study is the technique used to identify M2M traffic relative to standard traffic. The authors use the Type Allocation Code, which determines the kind of device that is generating the traffic. By consulting the GSM allocation database, one can determine whether the traffic is M2M or rather smartphone based. M2M devices are identified relative to an AT&T classification, complemented by public information, such as brochures and device specifications. The authors also divide the M2M devices in 6 categories, e.g., asset tracking metering, tele-health.

- Uplink vs. downlink. The first finding shows that M2M devices have a much larger uplink volume that downlink volume, in relative terms, compared to smartphones. This suggests a considerably different use of the network, and shows that M2M devices act more as content producers than consumers. There are also differences among the different categories of M2M traffic. At the same time, total M2M traffic, at least at the time of the study, was much lower that for smartphones. This situation may reverse as the number of M2M devices increases much more rapidly than smartphones. Network providers will have then to support a large number of low volume devices.

- Frequency patterns. The analysis also shows that M2M traffic follows business hours, and is significantly reduced in the weekends, as opposed to smartphone traffic which is virtually unchanged. Spectral analysis indicates strong periodicity in M2M traffic, corresponding to time intervals such as 1 hour, 30 minutes or 15 minutes, suggesting the timer-driven nature of M2M devices. Further analysis also reveals that devices are synchronized and coordinated. This may create congestion in the infrastructure. This frequency components are essentially absent from smartphone traffic.

- Sessions. The data shows that M2M devices have a lower active time than smartphone traffic. The length of a session, on the other hand, critically depends on the category of M2M device, with smartphone traffic in the middle. Sessions inter-arrival times are, instead, on average much longer for M2M devices than for smartphone traffic. M2M devices are active for traffic for much less time than smartphones. M2M traffic sessions occur much less frequently.

- Round Trip Time (RTT) and packet loss. The study looks at the time between the SYNC and ACK packets when establishing a TCP connection. The data shows that smartphone traffic

exhibits a lower RTT than M2M traffic. This is likely due to the used technology: smartphones are typically equipped with the latest network access technology, whereas M2M devices often still rely on 2G, which has longer delays. Among the M2M devices, telehealth is found to have the lowest RTT. Similarly, smartphone traffic and telehealth exhibit a lower packet loss rate. In some cases, packet loss may be due to the poor location of the devices (e.g., inside buildings).

One limitation with behavioral classification is that it generally classifies flows rather than packets. In this sense, the initial packets of the flow are of unknown nature, since not enough data has been yet received to obtain a meaningful result from the statistical analysis. This makes these techniques less useful in the context of a load balancing application. At the same time, port spoofing and encryption hamper the classification using DPI techniques. For this reason, a combination of DPI and DFI using statistical methods could provide the best results.

## 1.2 Overview

In this thesis, we are interested in particular in distinguishing between traditional user traffic, such as e-mail, web surfing and media streaming, from traffic originating from IoT or other machine to machine communications. We divided the work in two main analysis:

- Distinguish IoT flows from NON_IoT through the analysis of statistical features of the packets.
- Distinguish IoT flows from NON_IoT through the analysis of the frequency features of the packets.

To construct a classification procedure we use machine learning algorithms that can estimate the parameters for recognition from available labeled data. We therefore need to solve four problems:

- Generate or collect sufficiently comprehensive data for initial training and analysis.
- Develop an appropriate algorithm for classification using the given data.
- Have a good balance of IoT flow compared to NON_IoT.
- Choose the best features to construct the temporal series.

In our case, we have used several different techniques for generating flows of packets, from real IoT deployments to simulated systems, to have variety. In particular, we have seen different types of platforms, IBM-Bluemix [2], Node-RED [7], Contiki [1], and MIMIC Simulator [5]. Regarding the flows coming from real environments we will see all the types and the sources [48, 47, 49, 52, 51, 15, 16, 20, 12, 18] of the flow involved in our analysis. For the classification algorithm we have used the Weka framework [9], which provides the most popular machine learning methods out of the box and facilitates the interface between the data and the algorithm. In particular, we rely on two types of decision trees algorithms. The first is random forest algorithm, it has been shown to perform very well for the classification of the flows through the Fast Fourier Transform [53]. The second one is the J48 decision tree algorithm, a derivative of C4.5 [25], since it has been shown too, to perform well for internet flow classification [31]. We also explore clustering methods, using the k-means algorithm, to determine whether there is intrinsic structure in the data that can set apart the behavior of IoT devices from traditional communication. Our results show that this is only partly the case, and that the decision tree can provide better performance when devices of different kinds are employed. For the first type of analysis, we also evaluate the performance of other learning algorithms, such as neural networks and support vector machines. Several features, which are collected using libpcap library and the Tstat flow analysis software [34], are used for the classification. We deliberately ignore the port information, which could considerably simplify the problem. Instead, we focus more on the "behavioral" parameters, which are more independent of the protocols and robust to encryption. We

analyze two types of behavioral parameters, the packets frequency and the packets structure.

# 2 IoT Flow Generation tools

As discussed in the introduction, we have used several techniques to generate flows of IoT communication, including simulations, deployments and collecting data made publicly available on the Internet. In this section we discuss the different methods for traffic generation.

## 2.1 IBM-Bluemix - Node Red

IBM Bluemix is a platform that provides a service that makes several functionalities related to IoT applications available on the cloud [2]. One of the services offered by IBM Bluemix is Watson IoT Platform, which allows us to create applications, visualize a control dashboard, and simulate data exchange based on the MQTT protocol [6]. Alongside this, Node-RED is a flow-based programming tool for the IoT, which allows sensors and actuators to be connected to each other, as well as to APIs and online services required for their operation. It consists of a graphical platform which provides a palette of components which simulate the devices, and in which it is possible to connect the required peripherals. Taken together, Watson IoT Platform and Node-RED make it possible to build a complete system based on MQTT communication.

The flow editor will then show the system depicted in Figure 2.1.

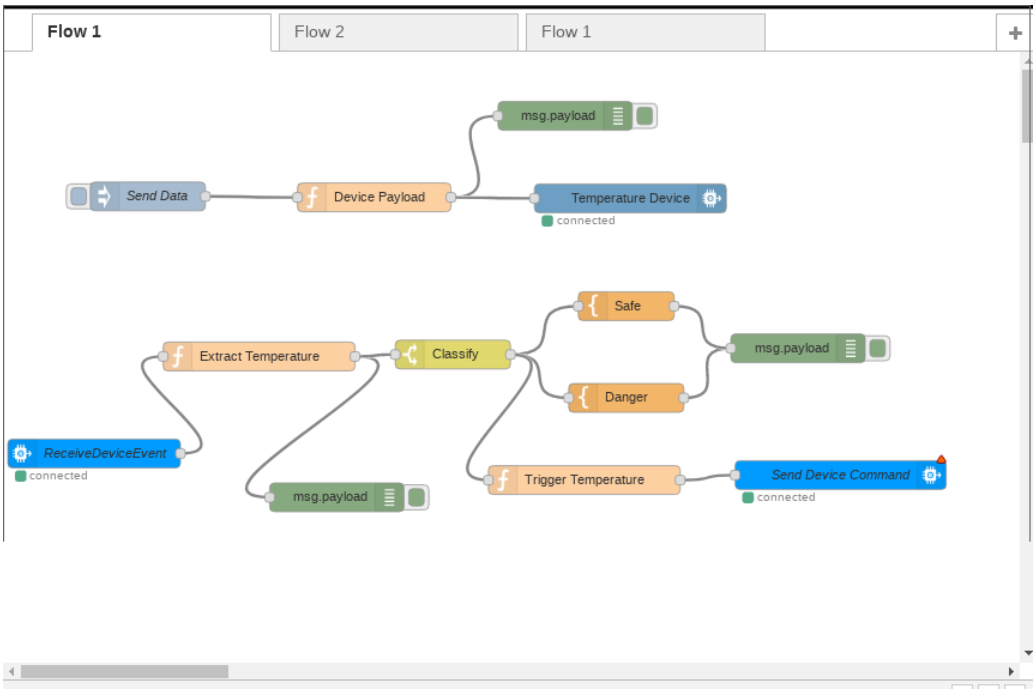



Figure 2.1: Screenshot of the temperature reading system in Node-RED flow editor.

Figure 2.2 gives a general overview of the relation between the different parts of the system.

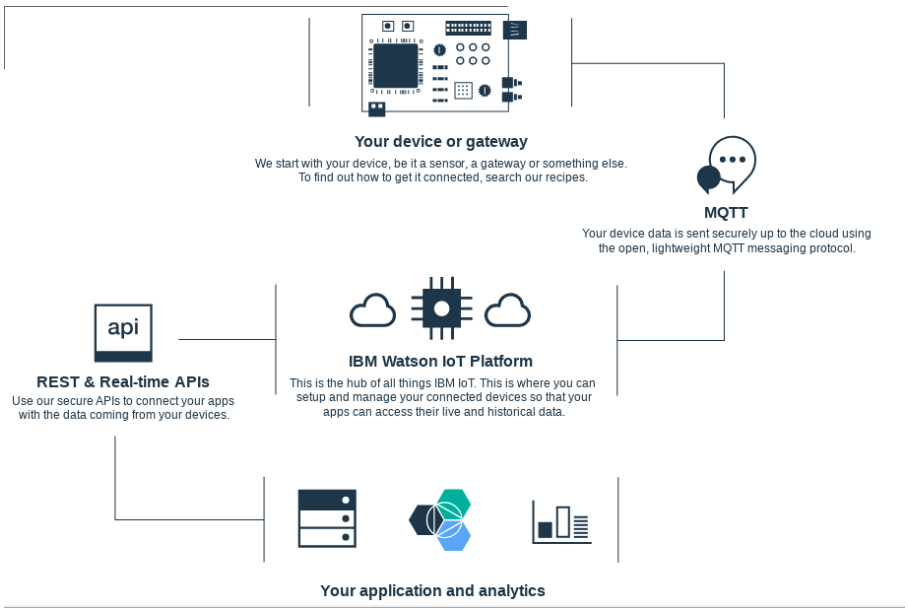



Figure 2.2: Connections around the IBM Watson IoT Platform.

## 2.2 Contiki

Contiki [1] is an open source, highly portable and light-weight multi-tasking operating system. It is written in the C language, and is particularly suitable for embedded systems with limited hardware resources. A typical Contiki configuration comprises 2 kilobytes of RAM and 40 kilobytes for ROM for the OS, and it was designed especially to extend the traditional Internet paradigm to the Internet of Things. We have used Contiki to generate CoAP packets. To do this, we will install the Contiki operating system on a virtual machine, and use the Cooja simulator to instantiate some simple nodes and have them communicate.

## 2.3 MIMIC Simulator

MIMIC Simulator [5] is able to create several virtual sensors capable of generating MQTT traffic, to test a system configuration before deployment. This is a commercial tool, however a trial version is more readily available. The trial lasts for 30 days, and allows one to simulate up to 25 concurrent sensors. Subsequently, we requested a more advanced trial, with the ability to simulate 250 sensors with a 15-day duration. Figure 2.3 shows the simulator main window. To simulate the sensors, click on "Add" on the main toolbar. A window will open which allows one to select the number of sensors that must be created in a range (e.g., from 1 to 25). In addition, we must select a valid private IP address within the subnetwork (e.g., 192.168.1.1/24). Finally, one can select the desired sensor to be simulated in the device section. From the menu, select Advanced, and in the Protocol section make sure that the MQTT box is checked. Figures 2.4a and 2.4b show the screenshots related to these operations.

Having done this, you can go into the newly created MQTT tab. In the first place, one needs to provide a protocol configuration file for the definition of the payload. There are several default that are already provided, which can generate various kinds of dynamic payloads. An example of configuration file is the following:

```
# Sample MQTT config file
# Copyright (c) 2016 Gambit Communications, Inc.
# this demonstrates dynamic messages generated from the same sensor
message = {
        topic = some-topic
        # in msec
        interval = 5000
        # QOS default 0
        # qos = 0
```

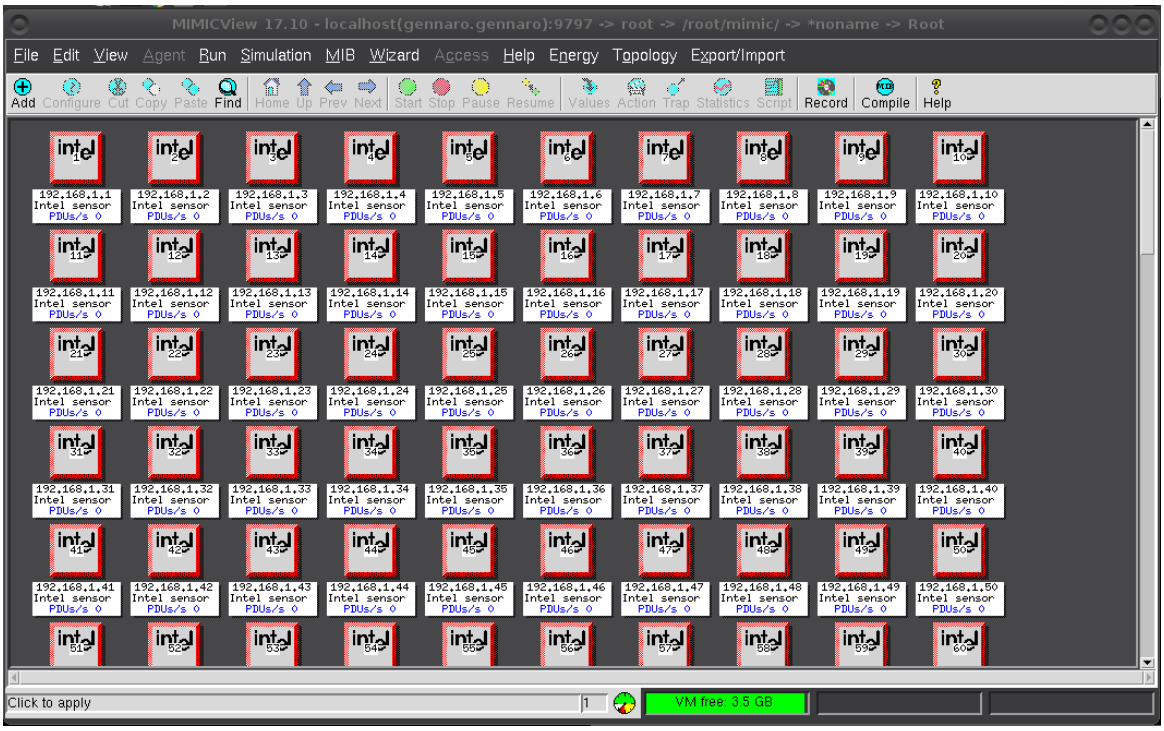
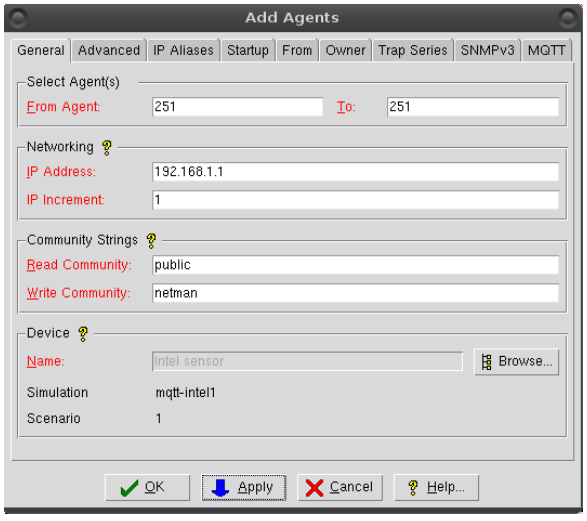
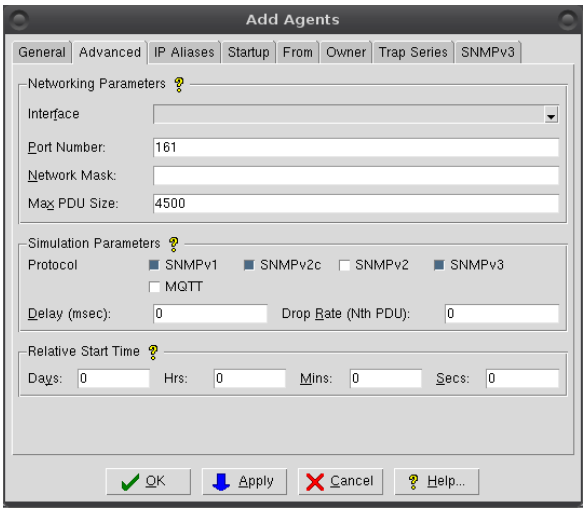



Figure 2.3: MIMIC main window.




(a) Selection of the IP address.




(b) Protocol selection.

```
        data = {
                # payload is dynamically generated:
                action = action1.mtcl
        }
}

message = {
        # dynamically generated topic for each agent of the form
        # /sensor/IP-ADDR/topic1 where IP-ADDR is the IP address
        # of the agent
        topic = action-topic-dynamic.mtcl
        interval = 10000
        qos = 1
        data = {
                # payload is dynamically generated:
                action = action2.mtcl
        }
}
```

The payload can be generated dynamically, while the time interval can be specified directly, together with the topic. We used different configuration files, to try to make the network as diverse as possible. It is possible to choose several brokers to which messages are sent. In the following section we will discuss the brokers that we have used for our analysis. Figure 2.5 shows a screenshot of the configuration and broker selection window.

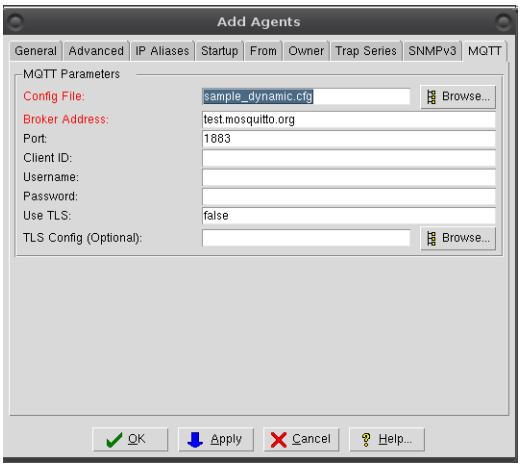



Figure 2.5: Configuration and Broker selection.

## 2.4 MQTT-Broker

In an MQTT system, the broker is responsible for receiving data from sensors and delivering information to the clients that request access. The broker follows a publish and subscribe paradigm, where sensors publish the information to the broker, which then alerts all the clients which have subscribed to the corresponding data. In our experiments, we have experimented with several different brokers, listed in Table 2.1.

In particular, we have used the HiveMQ MQTT broker at http://broker.hivemq.com, which then redirects to http://www.mqtt-dashboard.com. The advantage of this broker is that it does not limit the number of connections, as most of the other services do. The mosquitto broker also limits the number of connections, however it is simple to install a copy on a local personal computer.

### 2.4.1 Mosquitto

Mosquitto is an open source MQTT broker, and can be installed locally on the PC. Mosquitto is able to both send messages to another broker on the net, and to receive them. Mosquitto is very simple to use. In order to get more information, it is useful to change the configuration file so that we log more of the events. It is useful to try a simple publish and subscribe test, to verify that the installation was

Table 2.1: List of MQTT brokers employed in experiments.

| Port | Address | Note |
|---|---|---|
| 1883 | iot.eclipse.org | - |
| 1883 | test.mosquitto.org | - |
| 1883 | broker.hivemq.com | - |
| 18443 | cloudmqtt.com | Requires registration, limited number of connections |
| 1883 | mqtt.dioty.co | Requires registration |
| 1883 | mqtt.swifitch.cz | - |

successful. From two different terminals, we can first subscribe on a topic (for instance we can use hello/world), and then, from the other terminal, publish some text.

# 3 Capture and analysis tools

Several tools have been used to capture the raw data, organize it in flows, compute statistics and generate and evaluate classification algorithms. This section discusses them in turn.

## 3.1 Libpcap

Libpcap [3] is the library where all the analysis tools used in this work are based. It provides several API with the aim to capture and analyse the packets. To use libpcap in your program you have to include <pcap.h> in your code. Once the .pcap file is injected in our program through the function pcap_open_offline("file.pcap", error_buffer), we filter it, using the functions that libpcap provides. To filter the packets we use the function pcap_compile(), it takes in input an expression containing the filtering instructions, for instance, which types of packet we want discard or maintain. We use this string to filter the packets "tcp[tcpflags] & tcp-syn == tcp-syn and tcp[tcpflags] & tcp-ack != tcp-ack". This expression allow us to maintain the syn packets, discard all the syn acknowledge packets and obviously discard all the other types of packets. We apply this kind of filter because we want isolate the flows, so as to know their IP address. Once we have all the ip address, we filter another time the file. This time we use this string "(src port and dst port and src host and dst host) or (src port and dst port and src host and dst host)", so we can filter all the packets for each flow. Subsequently we analyze the entire flow with the function pcap_loop() that allow us to scan the packets and collect informations. Another useful function that libpcap provide us, is, pcap_open_live(), it allows us to capture packets from live network. Inside the program we have to define the packet headers, which contains various informations of the packet layers. We show a public piece of code named "Sniffex.c" [8] that contains the Ethernet header, the IP header and the TCP header.

```
/* Ethernet header */
struct sniff_ethernet {
        u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address */
        u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
        u_short ether_type;                     /* IP? ARP? RARP? etc */
};

/* IP header */
struct sniff_ip {
        u_char  ip_vhl;                 /* version << 4 | header length >> 2 */
        u_char  ip_tos;                 /* type of service */
        u_short ip_len;                 /* total length */
```

```
        u_short ip_id;                  /* identification */
        u_short ip_off;                 /* fragment offset field */
        #define IP_RF 0x8000            /* reserved fragment flag */
        #define IP_DF 0x4000            /* dont fragment flag */
        #define IP_MF 0x2000            /* more fragments flag */
        #define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
};
#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

/* TCP header */
typedef u_int tcp_seq;

struct sniff_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};
```

This piece of code contains all the informations provided by the ISO/OSI model. Regarding the datalink layer we have three fields, respectively, the source MAC address, the destination MAC address and the type of packet protocol. About the Network layer, we find, the destination and the source IP address, the total length of the packet, the type of protocol, the checksum and various flags. Finally we have the TCP header, we find the ports, the number of acknowledge and different types of flags. To this piece of code we added our code to compute di analysis to extract the flows. Here we can see the main function and how it works.

```
int main(int argc, char *argv[]) {

 a = fopen ("flussi.csv","w");

 char *device; /* Name of device (e.g. eth0, wlan0) */
 char error_buffer[PCAP_ERRBUF_SIZE];
 pcap_t *handle = pcap_open_offline(argv[1], error_buffer);

 u_char *my_arguments = NULL;
 struct bpf_program fp; /* The compiled filter expression */
```

```
char filter_exp[] = "tcp[tcpflags] & tcp-syn ==
tcp-syn and tcp[tcpflags] & tcp-ack != tcp-ack"; /* The filter expression */

bpf_u_int32 net; /* The IP of our sniffing device */
const u_char *packet; /* The actual packet */
struct pcap_pkthdr header; /* The header that pcap gives us */

if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
  fprintf(stderr, "Couldn't parse filter %s: %s\n",
  filter_exp, pcap_geterr(handle));
  return(2);
}
if (pcap_setfilter(handle, &fp) == -1) {
   fprintf(stderr, "Couldn't install filter %s: %s\n",
   filter_exp, pcap_geterr(handle));
   return(2);
}

pcap_loop(handle, -1, got_packet, my_arguments);
```

First we take the .pcap file as input, then we filter it with the function pcap_compile and then we call pcap_loop to scan the filtered file. As described previously the string "tcp[tcpflags] & tcp-syn == tcp-syn and tcp[tcpflags] & tcp-ack != tcp-ack" allow us to to maintain all the syn packets in such a way to detect and isolate the flows. After that we save the source and destination IP address and the respectively ports. After this preliminary scan phase, we provide to scan each flow, in order to save the payloads and the inter-arrival time to compute the Fast Fourier transform.

```
for(i = 0; i < number_of_flows; i++) {
    pcap_loop(handle, -1, got_packet, my_arguments);

}
//function called by pcap_loop
void got_packet(u_char *args, const struct
pcap_pkthdr *header, const u_char *packet){

if ( block != 0 && restart == 1 &&
   tcp->th_flags != 2 ) {

   int re = strcmp(save_ip[0],inet_ntoa(ip->ip_src))
      if ( re == 0 ) {
    long prima = NapTime;
            long prima_sec = NapTime_sec;
            NapTime = (header->ts.tv_usec);
    NapTime_sec = (header->ts.tv_sec);
            if (packet_counter == 0) {
     flow_sequence[packet_counter] = size_payload;
             flow_sequence_TIME[packet_counter] = 0 ;

    } else {
     long seconds = NapTime_sec - prima_sec;
     long micro_seconds = NapTime - prima;

     if ( micro_seconds < 0 ) { seconds -= 1;}
```

```
        long total_micro_seconds = (seconds *
                1000000) + abs(micro_seconds);
                flow_sequence_TIME[packet_counter] =
                total_micro_seconds;
        flow_sequence[packet_counter] =
                size_payload ;
       }
                packet_counter++;
      } else {
        if(packet_counter_inv < 256) {
         flow_sequence_INVERSE[packet_counter] =
                 size_payload;
packet_counter_inv++;
        }
            }
      if ( ( tcp->th_flags == 17 )  ||
         ( packet_counter == 256 )  ||
         ( ( packet_counter_inv == 256 )  &&
         ( packet_counter > 128 ) ) ) {
//Compute FFT
          }

}
```

First of all, iteratively we scan each flow calling the function got_packet invocated by pcap_loop. Inside the function got_packet we made our analysis, initially we save the source MAC address, it is used to recognize and to label each device. Then we save in three arrays the payloads of the packets, respectively from client to server, from server to client and the inter-arrival times from client to server. Finally, we have the block of conditions, inside them we compute the FFT from the series. We stop the scan of the packets, when we see a FIN packet, this means that the flow is finished, or when we see 256 packets from client to server or when we have 256 packets from server to client and 128 from client to server. If one of this three condition is verified, we save the frequency spectrum, we label the flow and we save it in a csv file.

## 3.2 Sniffer: Wireshark and tcpdump

To capture the packets generated with the above methods we need special software that can monitor the activity on the network interfaces and record all the packets that are transmitted and received. We can do this using the Wireshark sniffing software, which is free, easy to use, well documented and actively maintained. Wireshark provides a graphical user interface with which to observe the network traffic, select the desired network interface, and capture the packets in real time. The software presents both the raw data in the form of a hexadecimal dump, as well as the dissected information regarding the various levels of the protocols that are used in the communication, including the source and destination IP addresses and ports. In particular, Wireshark understands most of the IoT related protocols. Once the data is captured, it can be exported to files that can be used for off-line analysis, whether it is deep packet inspection or flow inspection. Several file formats are supported (e.g., pcap, pcapng, csv, raw). Integrated filters can be used to select exactly the kinds of packets that need to be exported, discriminating on the protocols, the addresses, and so on. In figure 3.1 we illustrate how a packet is visualized in wireshark. Besides Wireshark, we have also used tcpdump, a more lightweight sniffer without a graphical user interface. The software is actually rather powerful, and implements a set of filters to select the packets or hosts on which to intercept the traffic. Running from the command line makes it extremely useful in scripting applications. In addition, the file formats are compatible with Wireshark.

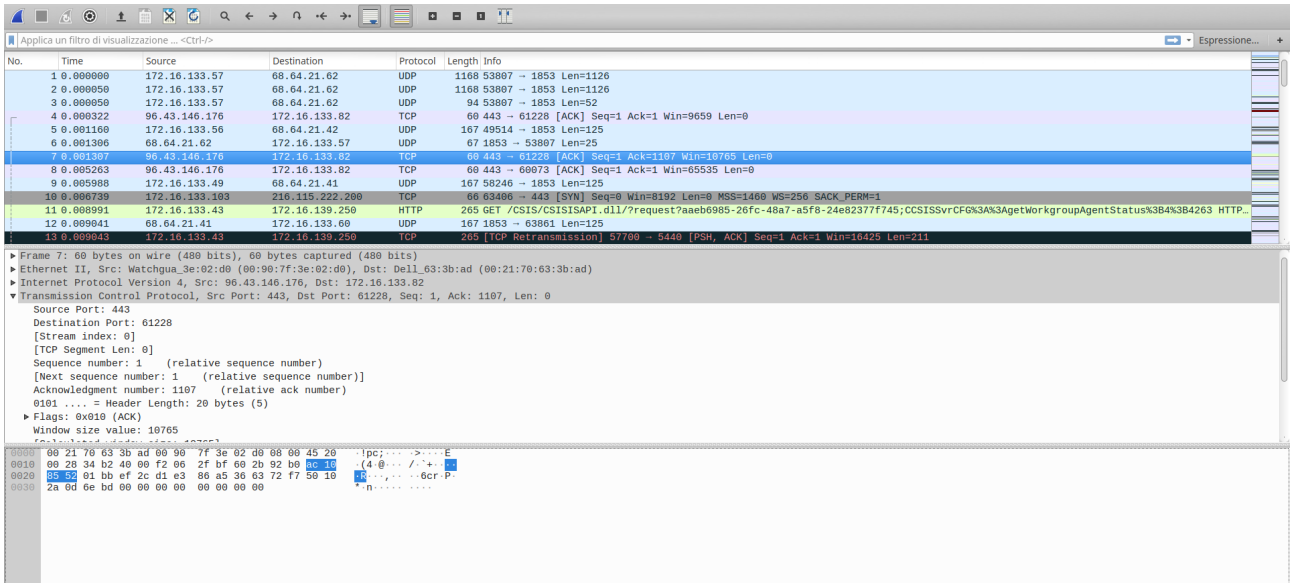



Figure 3.1: Wireshark Screenshot

## 3.3 Tstat

Tstat is a tool that is able to analyze network traffic and provide useful information on the performance indexes and give statistical information on the communication parameters. Tstat can generate statistics in two ways: by capturing packets directly from the network interface (using the same libraries as tcpdump), or analyzing packet traffic traces that were previously captured and saved on a file. Tstat supports several formats, and is compatible with the traffic capture software described in the previous section.

We use Tstat primarily to analyze previously captured data, we will therefore invoke the software using the following command:

```
tstat -g -H histogram.txt dump_file
```

where `-g` instructs the software to generate global statistical information (see below), and `-H histogram.txt` determines which statistics to collect. We are generally interested in all the available data, so that `histogram.txt` contains the simple line

```
include ALL
```

Tstat generates two kinds of output. The first is a set of log files which contain information about the connections that were detected during the communication. Tstat understands the concept of "flow", i.e., a series of packet exchanges related to the same communication, identified by source and destination IP addresses and ports, and combines the data from different packets that belong to the same flow. When using the TCP transport protocol, a flow is identified from the moment it is opened using the SYN - SYN/ACK sequence, to the moment it is closed using FIN/ACK or RST sequence. A flow is also closed whenever no packet is observed related to the flow for a default duration of 10 seconds, or if more than 5 minutes have elapsed from the last packet in general.

More in detail, the log files which are generated as part of the analysis are the following:

- "log_tcp_complete" and "log_tcp_nocomplete": in these two files all the information related to TCP flows is recorded. The "complete" log file contains information on all the correctly identified TCP flows, while the "nocomplete" log file lists information on those flows which have not been correctly identified. This may happen, for instance, if the packet capture was started after the connection was already established, or before it is closed.

- "log_udp_complete": similarly to TCP, this file reports information on the UDP flows. A UDP flow is identified whenever the first segment of the protocol is observed for a pair of UDP ports and IP addresses, and terminates when no packets are observed for 10 seconds from the beginning of the connection, or after 3 minutes and 20 seconds have elapsed from the last packet in general.

- "log_video_complete": this file tracks all the video TCP flows. The connections are subdivided into RTMP and TLS, which are associated for instance to youtube, and HTTP connections.
- "log_http_complete": this log file tracks information related to all the HTTP requests and responses. This file is not generated by default, but must be requested when the software is invoked.
- "log_mm_complete": reports statistics on multimedia flows, such as RTP and RTCP flows.
- "log_skype_complete": reports statistics related to Skype traffic.
- "log_chat_complete, log_chat_messages": reports statistics related to messaging applications and chats in general.

Within each log file we find the values related to the corresponding kind of communication. The file is organized in columns, which are grouped according to the direction of traffic, from client to server or from server to client. The reported information is very comprehensive, and is shown on Figure 3.2, taken from the Tstat documentation. In the first place, we find the IP addresses of the client and

| C2S | S2C | Short description | Unit | Long description |
|---|---|---|---|---|
| 1 | 15 | Client/Server IP addr | - | IP addresses of the client/server |
| 2 | 16 | Client/Server TCP port | - | TCP port addresses for the client/server |
| 3 | 17 | packets | - | total number of packets observed form the client/server |
| 4 | 18 | RST sent | 0/1 | 0 = no RST segment has been sent by the client/server |
| 5 | 19 | ACK sent | - | number of segments with the ACK field set to 1 |
| 6 | 20 | PURE ACK sent | - | number of segments with ACK field set to 1 and no data |
| 7 | 21 | unique bytes | bytes | number of bytes sent in the payload |
| 8 | 22 | data pkts | - | number of segments with payload |
| 9 | 23 | data bytes | bytes | number of bytes transmitted in the payload, including retransmissions |
| 10 | 24 | rexmit pkts | - | number of retransmitted segments |
| 11 | 25 | rexmit bytes | bytes | number of retransmitted bytes |
| 12 | 26 | out seq pkts | - | number of segments observed out of sequence |
| 13 | 27 | SYN count | - | number of SYN segments observed (including rtx) |
| 14 | 28 | FIN count | - | number of FIN segments observed (including rtx) |
| 29 | | First time abs | ms | Flow first packet absolute time (epoch) |
| 30 | | Last time abs | ms | Flow last segment absolute time (epoch) |
| 31 | | Completion time | ms | Flow duration since first packet to last packet |
| 32 | | C first payload | ms | Client first segment with payload since the first flow segment |
| 33 | | S first payload | ms | Server first segment with payload since the first flow segment |
| 34 | | C last payload | ms | Client last segment with payload since the first flow segment |
| 35 | | S last payload | ms | Server last segment with payload since the first flow segment |
| 36 | | C first ack | ms | Client first ACK segment (without SYN) since the first flow segment |
| 37 | | S first ack | ms | Server first ACK segment (without SYN) since the first flow segment |
| 38 | | C Internal | 0/1 | 1 = client has internal IP, 0 = client has external IP |
| 39 | | S Internal | 0/1 | 1 = server has internal IP, 0 = server has external IP |
| 40 | | C anonymized | 0/1 | 1 = client IP is CryptoPAn anonymized |
| 41 | | S anonymized | 0/1 | 1 = server IP is CryptoPAn anonymized |
| 42 | | Connection type | - | Bitmap stating the connection type as identified by TCPL7 inspection engine (see protocol.h) |
| 43 | | P2P type | - | Type of P2P protocol, as identified by the IPP2P engine (see ipp2p_tstat.h) |
| 44 | | HTTP type | - | For HTTP flows, the identified Web2.0 content (see the http_content enum in struct.h) |

Figure 3.2: Data reported in the TCP log file.

the server, the ports that are used, the total number of packets, and the number of RTS and ACK segments that were sent. We then have information about the size of the transmission, with the total bytes and number of packets that were exchanged. Finally there is timing information, giving a measure of the overall flow duration and for certain specific packets.

An additional functionality is the ability to generate histograms related to the captured data. A histogram represents the distribution of a specific performance index, considering a fixed measurement period. For this functionality, Tstat saves numerous files corresponding to the statistics of the flows observed over intervals of 5 minutes. Traffic is again subdivided according to its direction, as before. Several parameters can be analyzed:

- IP Layer: statistics related to the IP addresses and the protocols that were used.
- TCP Segments: statistics related to the TCP segments.

- TCP Flows: statistics related to the TCP flows.
- UDP Layer: statistics related to the UDP flows.
- Streaming Flows: statistics related to flows that carry streaming data.
- RTCP Flows: statistics related to the RTCP protocol.
- HTTP Flows: statistics related to the HTTP protocol.
- Profile: profile of the computer on which Tstat is executing.

## 3.4 Weka

Weka is a Java-based analysis software which collects a large number of machine learning algorithms and analysis methods to be applied to sets of data. Weka provides a generic interface with which to solve classification problems, regressions, clustering, association rule extraction and is capable of generating diagrams and plots. In addition to the graphical user interface, Weka can also be used directly through the Java language. One way to use Weka is to apply the machine learning algorithms to the dataset for analyze the data in more detail, or to use models to predict the results.

Running Weka is particularly simple, once the installation files have been downloaded. Simply go into the download directory, and run

```
java -jar weka.jar
```

The simplest method to use Weka is through its own graphical user interface, called "Explorer", which can be selected from the Applications menu. In the Explorer, one can select among six different panels. In the first, Preprocess, it is possible to select and load a dataset, which is a text file in the Attribute Relationship File Format ("arff"). The `arff` file is made of two sections: the header and the actual data. In the first line of the header we find the name of the relation, followed by a list of attributes, which represent the kind of data found for each record, with their names and type. There can be several types of attributes:

- Numeric: it is a numeric type represented by a floating point value.
- Nominal: it is an enumerated type, with a predefined number of values.
- String: a string, typically used in text classification.
- Date: represents a data, entirely described in floating point.
- Relational: it is a kind of attribute that may contain other attributes.

Finally, the header contains the classes to which the records which are listed in the subsequent section belong. This is typically a nominal type, i.e., an enumerated list of possible alternatives. This is a key parameter in classification, since it provides the resolution by which the algorithms operate. After the header, the file contains the data, described sequentially. Each attribute is separated from the next by a comma, and the record is terminated by its class. A simplified example of arff file that we have used in our tests is the following:

```
@relation 'cccc-weka.filters' @attribute c_pkts_all:3
numeric @attribute c_bytes_all:9 numeric @attribute
s_pkts_all:17 numeric @attribute s_bytes_all:23 numeric
@attribute durat:31 numeric @attribute
inter-arrival-time numeric @attribute c_rtt_avg:45
numeric @attribute c_rtt_min:46 numeric @attribute
Protocol {TCP,VPN,MQTT,CASA} @data
5,572,5,296,607.911,60.7911,0.269658,0.198,TCP
5,354,5,1193,282.466,28.2466,1.622613,0.324,VPN
5,490,5,296,602.203,60.2203,0.31199,0.219,TCP
5,354,5,1193,333.994,33.3994,0.31199,0.239,MQTT
```

```
5,612,5,296,578.331,57.8331,0.272991,0.215,TCP
5,572,5,296,644.661,64.4661,0.615646,0.189,TCP
5,354,5,1193,300.728,30.0728,0.254658,0.148,TCP
5,599,5,296,586.181,58.6181,0.253325,0.211,CASA
5,354,5,1193,325.747,32.5747,0.278991,0.182,TCP
```

As shown, the data is listed in order for each record and correspond to the attributes in the header, terminated by the Protocol class. Once the file is loaded, Weka creates a histogram which graphically shows the distribution of the attributes and their class. More in detail, the histogram shows the number of times that an element of a certain class (classes are color coded) has an attribute in a certain range. This could be useful to visually determine if an attribute is able to discriminate among the various classes. Figure 3.3 shows the main Weka window for the complete file.

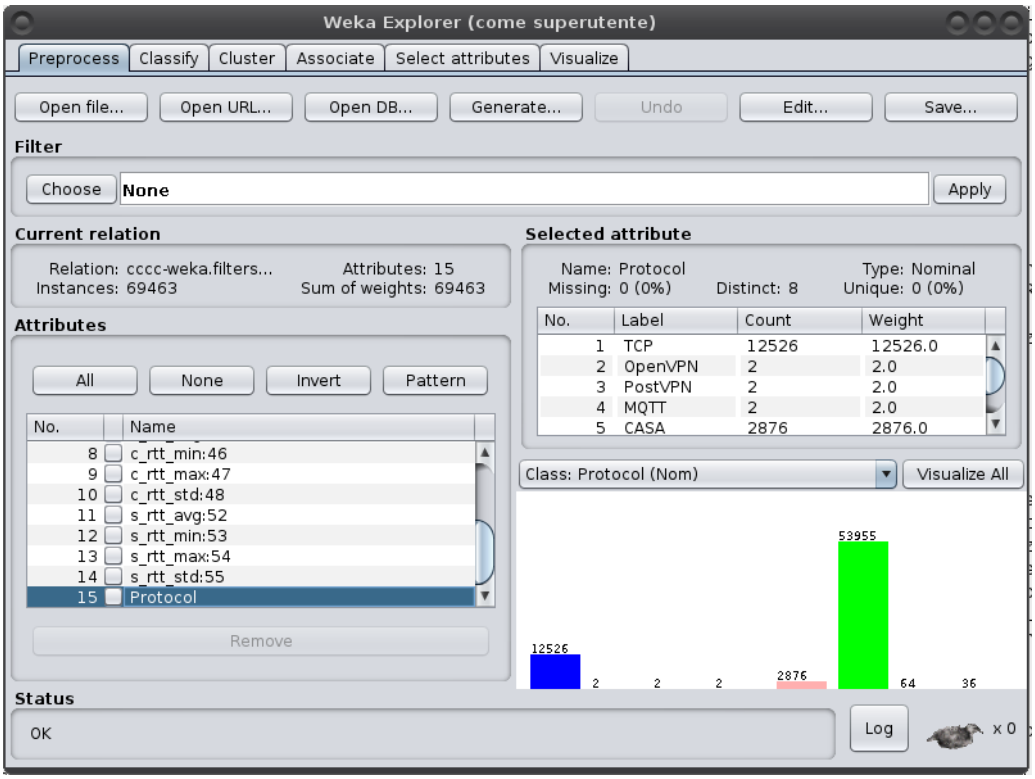



Figure 3.3: Weka Explorer window, showing attributes and classes.

It is also possible to filter the data for use with different kind of learning algorithms, such as supervised and non-supervised. There are several filters. For instance, one can normalize the data or discretize them by dividing them into bins of defined size.

Weka provides several learning algorithms, which can be selected in the "Classify" tab. Among them:

- Various instances of Bayesian classifiers;
- Functions, including linear regression and logistic regression, as well as neural networks (multilayer perceptron) and support vector machines;
- Rules, such as the basic ZeroR, OneR, and algorithms such as DecisionTable, JRip, and PART;
- Trees, such as HoeffdingTree and J48;

Once the algorithm has been chosen, we can select whether the previously loaded dataset should be used entirely as a training set, or if we want to split it into folds, or again if we want to use only a fraction of it. The training set is that part of the data which is used for learning, while the validation set is the part used to validate and test the model that is created by the learning algorithm.

Once the test options have been selected, we can execute the learning algorithm by clicking on Start, which will train the classifier. For our example, we have used the J48 decision tree algorithm. The output is structured as follows: first we find the list of attributes, and how these have been used in the tree; then we find the actual tree, showing which parameters have been used as path discriminators, and their value; finally, the system provides a summary that shows the statistics regarding the performance of the classification algorithm. For our example (using the non-simplified version):

```
 Correctly Classified Instances
12048               96.1379 % Incorrectly Classified Instances         484
3.8621 % Kappa statistic                          0.9608 Mean absolute error
0.0012 Root mean squared error                  0.0248 Relative absolute error
6.2703 % Root relative squared error             25.0411 % Total Number of
Instances               12532
```

In the first place, we find the number of records that were correctly classified, and those that we incorrectly classified, with their corresponding percentage. More statistics include the degree of accuracy and reliability, error information such as the root mean squared error and relative absolute error, and the total number of instances (records) that were analyzed. The output is then integrated with more accuracy details regarding the various classes. It consists of 9 columns and as many rows as the number of classes. For our example, we obtain:

```
 TP Rate  FP Rate  Precision  Recall
F-Measure  0,950    0,000    0,938      0,950    0,944      0,942    0,000
0,970      0,942    0,956    0,982    0,001    0,974      0,982    0,978
0,966    0,001    0,957      0,966    0,961

MCC      ROC Area  PRC Area  Class 0,944    1,000     0,981     cluster0 0,955
1,000     0,981     cluster1 0,978    1,000     0,996     cluster2 0,961
1,000     0,986     cluster3
```

Le columns report, in order, the rate of True Positives (TP), of False Positives (FP), the precision, the recall and the F-Measure which combines the previous metrics. In addition, it reports the Matthews Correlation Coefficient (MCC), the Receiver Operatin Characteristic (ROC), the PRC and finally the class. Finally, the results include the *confusion matrix*: each column represents the predicted values, while the rows represent the real values, for each class. Non-zero elements outside the diagonal correspond to classification errors. An example is shown in Figure 3.4. The matrix gives an indication

```
=== Confusion Matrix ===

    a     b     c     d     e     f     g     h   <-- classified as
12524     0     0     0     0     2     0     0 |     a = TCP
    0     2     0     0     0     0     0     0 |     b = OpenVPN
    0     0     2     0     0     0     0     0 |     c = PostVPN
    1     0     0     0     0     0     0     1 |     d = MQTT
    0     0     0     0  2872     4     0     0 |     e = CASA
    0     0     0     0     0 53955     0     0 |     f = CINA
    0     0     0     0     0     0    57     7 |     g = INTEL
    0     0     0     0     0     0     1    35 |     h = BOSCH
```

Figure 3.4: Confusion matrix for our example.

of how well elements are classified, and how the error is distributed across the various classes.

Another important section in the Weka tool is dedicated to clustering. In this case, the input is analyzed and records are divided in clusters according to their attributes. From the interface, one can select several clustering algorithms, including Canopy, Cobweb, EM, FarthestFirst, FilteredClustered, HiearchicalClusterer, MakeDensityBasedClusterer and SimpleKMeans. Also in the case of clustering, the dataset can be split between training and test data, according to a specific percentage. In our case, we have used the SimpleKMeans algorithm. The generated output includes a table which reports the centroid used for each cluster, and a second table that reports how many records (flows in our

case) have been assigned to a cluster rather than another. It is also possible to graphically visualize the cluster assignment and their distribution. The results can be exported, in a way that the clusters take up the function of a class.

## 3.5 KissFFT

To compute the Fast Fourier Transform (FFT) we used a specific library called, "KissFFT"[4]. It provides several functions to compute the fast fourier transform. Let we see a piece of code

```
#include "kiss_fft.h"
    kiss_fft_cfg cfg = kiss_fft_alloc( nfft ,is_inverse_fft ,0,0 );
    while ...

        ... // put kth sample in cx_in[k].r and cx_in[k].i

        kiss_fft( cfg , cx_in , cx_out );

        ... // transformed. DC is in cx_out[0].r and cx_out[0].i

    kiss_fft_free(cfg);
```

To use this library in our program we have to include the file "kiss_fft.h" in our code. The function kiss_fft() takes in input the FFT allocation parameters (it specifies the number of the fft points), the input series "cx_in", and the output "cx_out". To have the amplitude of the signal we have to compute the square root of sum of square of the real and the imaginary part.

```
for(i = 0; i < N / 2 + 1; i++) {
  mags[i] = hypotf(out[i].r,out[i].i);
}
```

# 4 Datasets

In this work, for both of our experiments we used a variety of public repository to obtain the pcap files, and from them then construct our datasets.

## 4.1 Pcap Resources

Most of pcap files used for the analysis comes from a public repository provided by University of New South Wales (UNSW Sydney), Australia [48, 49, 47].

They developed a small smart IoT environment with different types of devices, IoT and non_IoT, interconnected through a wireless network. From this resource we take 2 types of flows, the IoT flows from this devices:

Table 4.1: Australian IoT devices

| IoT Device | MAC address |
|---|---|
| Smart Things | d0:52:a8:00:67:5e |
| Amazon Echo | 44:65:0d:56:cc:d3 |
| Netatmo Welcome | 70:ee:50:18:34:43 |
| TP-Link Day Night Cloud camera | f4:f2:6d:93:51:f1 |
| Samsung SmartCam | 00:16:6c:ab:6b:88 |
| Dropcam | 30:8c:fb:2f:e4:b2 |
| Insteon Camera | 00:62:6e:51:27:2e |
| Insteon Camera | e8:ab:fa:19:de:4f |
| Withings Smart Baby Monitor | 00:24:e4:11:18:a8 |
| Belkin Wemo switch | ec:1a:59:79:f4:89 |
| TP-Link Smart plug | 50:c7:bf:00:56:39 |
| iHome | 74:c6:3b:29:d7:1d |
| Belkin wemo motion sensor | ec:1a:59:83:28:11 |
| NEST Protect smoke alarm | 18:b4:30:25:be:e4 |
| Netatmo weather station | 70:ee:50:03:b8:ac |
| Withings Smart scale | 00:24:e4:1b:6f:96 |
| Withings Aura smart sleep sensor | 00:24:e4:20:28:c6 |
| Light Bulbs LiFX Smart Bulb | d0:73:d5:01:83:08 |
| Triby Speaker | 18:b7:9e:02:20:44 |
| PIX-STAR Photo-frame | e0:76:d0:33:bb:85 |
| HP Printer | 70:5a:0f:e4:9b:c0 |
| Nest Dropcam | 30:8c:fb:b6:ea:45 |

and the NON_IoT flows from this devices (see table 4.2).

To have a variety, we introduced simulated flows generated by a software simulator, MIMIC MQTT simulator [5]. This software allows to simulate up to 250 concurrent sensors, divided in two main brands and configurations, Intel and Bosch. In our analisys the two types of sensors are put togheder in one class, defined "MIMIC". Using different configuration files, one can try to make the network as diverse as possible, simulating different sensors and periodicities. Nevertheless, the behaviors will be somewhat homogeneous, as it is difficult to model event-triggered sensors in this framework.
To have more IoT resources we introduced more Pcap files provided by the USC/LANDER project [52, 51]. These pcap files contains first-time boot-up traffic of multiple IoT devices located in a LAN network. Traces are captured at LAN port of the LAN router.

Table 4.2: Australian Non-IoT devices

| NON_IoT Devices | MAC address |
|---|---|
| Android Phone | 40:f3:08:ff:1e:da |
| Laptop | 74:2f:68:81:69:42 |
| MacBook | ac:bc:32:d4:6f:2f |
| Android Phone | b4:ce:f6:a7:a3:c2 |
| IPhone | d0:a6:37:df:a1:e1 |
| MacBook/Iphone | f4:5c:89:93:cc:85 |
| Samsung Galaxy Tab | 08:21:ef:3b:fc:e3 |

Let we see the devices in this table (see table 4.3).

Table 4.3: Californian IoT devices

| IoT Devices | MAC Address |
|---|---|
| HP Printer | 34:64:a9:8d:56:cb |
| TP-Link Smart plug | 50:c7:bf:09:08:44 |
| Amazon Dash Bounty Button | 50:f5:da:3d:fd:43 |
| Foscam IP CAM2 | 0c:84:dc:62:6d:5f |
| Amazon Echo | 68:37:e9:b3:ad:b1 |
| Google Smart Speaker | 30:fd:38:04:25:32 |
| Amazon Fire SmartTVStick | 18:74:2e:e3:4e:22 |
| Philips-Hue | 00:01:5c:69:82:47 |
| AMCREST IP CAM | 3c:ef:8c:8c:0d:c1 |
| RENPHO Humidifier | dc:4f:22:0f:f1:b3 |
| Belkin Wemo switch | 60:38:e0:f0:cb:61 |
| TENVIS IP cam | 14:6b:9c:a6:b2:7c |
| D-link IP CAM | b0:c5:54:22:b3:ba |
| TP-Link SmartLightBulb | 50:c7:bf:5f:02:b1 |
| Foscam IP CAM | a0:c9:a0:f8:65:49 |
| Wize IP CAM | 2c:aa:8e:02:ee:ba |

Now we introduce different pcap files containing NON_IoT flows. The first comes from an experiment from the University of New Brunswick [20]. They captured different real traffic from many types of services. They collect packets from web browsing, emails, chatting sessions, video streaming like youtube or netflix, file transfering, voip and p2p services. These flows are very usefull because rapresents the most of traffic we generate everyday. In table 4.4 we see the different types services they have collected.

Table 4.4: Canadian NON_IoT flows

| NON_IoT Services | NON_IoT Services |
|---|---|
| Facebook | Youtube |
| Email | Vimeo |
| Torrent | Spotify |
| File Transfer Protocol | Skype |
| Gmail | Scp |
| Hangouts | Netflix |

Then we introduced other flows coming from a repository collected by the Network Monitoring and Measurements research group at the University of Napoli [15, 16]. They collected traffic on port

80 generated by clients inside the network of University of Napoli, reaching the outside world. We also include the "bigFlows" traffic dataset from Appneta Tcpreplay [12]. This is a capture of real network traffic on a busy private network's access point to the Internet. Finally we obtained from an online repository in Japan [18] other NON_IoT flows dedicated to traffic anomaly detection.

# 5 Packet Statistical Analysis

This chapter takes inspiration of our publication [19]. For this first part of our analysis investigating on the fractions of the packets length we have followed two methodologies to develop a flow classification method. The first is based on clustering, while the second is based on general purpose classification algorithms, with particular focus to classification trees. We first present the dataset previously discussed in Chapter 4 used for this section of our experiments. Then we discuss the two approaches in detail.

## 5.1 Dataset preparation and Baseline classification

The dataset is shared between both classification methods. It consists of information collected using Tstat on approximately 77 thousand flows, which were later in the project complemented by an additional 15 thousand IoT flows collected from an online repository. The vast majority of IoT flows of the initial dataset are obtained by capturing the simulated IoT systems using the MIMIC software simulator. We have employed different configurations for the virtual sensors, in order to have some variability in both the communication interval and the size of the payload. Nonetheless, the generated traffic is rather homogeneous, as the classification results will show it. At the same time, we subscribe to the published data, so that the broker sends packets to the subscribers as soon as the data is published. All this traffic is classified as IoT. Additional IoT flows in the initial dataset were captured from a real IoT deployment in a city-wide environment in Trento, in collaboration with Create-Net. The system is composed of number of sensors deployed city-wide and a collection point over a LoRa protocol which then sends data to a server, which can be queried using MQTT. A few more flows are obtained directly using the methods discussed above. The IoT flows in this first set are, in total, 7,762.

Most of the non-IoT flows (around 54,000) are obtained from an online repository in Japan [18], dedicated to traffic anomaly detection. To complement these, roughly 3,000 non-IoT flows were captured in the domestic environment while running the MIMIC simulator, having extra applications run regular communication patterns, and from the Create-Net deployment, for roughly 12,500 flows, setting aside the IoT packets. The flows were manually labeled, as shown in Table 5.1.

Table 5.1: Flow manual categorization.

| **Class** | **Flows** | **Austr. flows** | **Total** |
|---|---|---|---|
| IoT | 7,762 | 15,081 | 22,843 |
| NON_IoT | 69,357 | – | 69,357 |
| Total | 77,119 | 22,843 | **92,200** |

A second set of 15,081 IoT flows was obtained from an online repository. Traffic is captured in a domestic environment from a real deployment in Australia, where a house was instrumented with several devices interconnected through a wireless network [49, 48, 47]. These flows are particularly relevant, since the range of devices is very diverse (see Table 5.2), and they come from a real deployment. These have been used in a second step, since they were not available earlier. For this reason, we first report the results obtained with the initial dataset, and then analyze how these change with the addition of more IoT flows. This also highlights the difference in clustering and classification accuracy

Table 5.2: Devices in the online Australian deployment.

| **Smart device** | **Smart device** |
|---|---|
| Amazon Echo | Netatmo Welcome |
| TP-Link Day Night Cloud camera | Samsung SmartCam |
| Dropcam | Insteon Camera |
| Withings Smart Baby Monitor | Belkin Wemo switch |
| TP-Link Smart plug | iHome |
| Belkin wemo motion sensor | NEST Protect smoke alarm |
| Netatmo weather station | Withings Smart scale |
| Blipcare Blood Pressure meter | Withings Aura smart sleep sensor |
| Light Bulbs LiFX Smart Bulb | Triby Speaker |
| PIX-STAR Photo-frame | HP Printer |
| Nest Dropcam | |

between a simulated and a real environment.

Once the flows have been captured, the statistics are collected using Tstat to generate the log files. For this application, we use only the overall information, and do not make use of the generated histograms. The log files must be pre-processed in order to be used with Weka. In particular, we must transform them into a "comma separated values" (`.csv`) format. For our application, we have then used a spread-sheet to compute additional parameters. For instance, we have computed the average inter-arrival time as the ratio between the number of packets and the flow duration. This information was then saved and added to the `.csv` file. The dataset can now be imported in Weka. In Weka, we can inspect the statistics file using the ArffViewer, which can be invoked from the main window under the Tools menu. We can then load the `.csv` that we have just created, and modify it to remove for instance attributes and/or flows that we do not intend to use during the classification. The file can then directly be saved in the `.arff` format from within the ArffViewer. The data is now usable by Weka.

Before exploring the classification methods, we report the baseline results that can be obtained using an extremely naïve approach which classifies all flows according to the most abundant class. This classification method is known as `ZeroR`. In our initial dataset (not including the flows from the Australia deployment), there are many more NON_IoT than IoT flows, so if we were to classify everything as NON_IoT, using our entire dataset for testing, we achieve a classification accuracy of:

$$A = \frac{\text{NON\_IoT flows}}{\text{Total flows}} = \frac{69357}{77119} = 89.9\%$$

A similar reasoning is applied to the complete dataset, which includes the flows from the Australian deployment, to obtain:

$$A = \frac{\text{NON\_IoT flows}}{\text{Total flows}} = \frac{69357}{92200} = 75.2\%$$

Because this accuracy is already rather high, we must be careful to consider different metrics to evaluate our classification strategies, and consider especially the accuracy with which IoT flows are classified. Besides the class specific True Positive (TP) and False Positive (FP) rate, and the precision and recall, one alternative measure, which is particularly significant for binary classification especially when the classes are of different size as in our case, is the *Matthews correlation coefficient* (MCC). The MCC returns a value between -1 and +1, where +1 represents a perfect prediction, 0 is no better than a random prediction and -1 indicates total disagreement between prediction and observation. For the initial simulated dataset, by applying the ZeroR classification method in Weka we obtain the performance parameters reported below:

```
=== ZeroR Summary ===

Correctly Classified Instances       69357               89.935  %
```

```
Incorrectly Classified Instances      7762               10.065  %
Kappa statistic                          0
Mean absolute error                      0.181
Root mean squared error                  0.3009
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances            77119

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.101     IoT
1.000    1.000    0.899      1.000    0.947      0.000    0.500     0.899     NON_IoT
0.899    0.899    0.809      0.899    0.852      0.000    0.500     0.819     Weighted Avg.

=== Confusion Matrix ===

     a     b   <-- classified as
     0  7762 |     a = IoT
     0 69357 |     b = NON_IoT
```

The report naturally shows that the precision for IoT flows is zero (the result is similar for the complete dataset). Similarly, the MCC is zero, indicating that the method performs in a limit case. The objective of our study is to maximize these values.

### 5.1.1 Attribute selection

The flow analysis with Tstat provides a large number of features, all of which are not necessarily relevant to our objectives. First, our aim is to be able to operate in the presence of encryption. For this reason, we assume that certain parameters, such as the IP addresses and the port numbers, are not visible to the application. This may happen, for instance, if the flows are carried along a VPN. It has been shown that these features are of extreme help in classification [49]. Thus, we set ourselves in a rather unfavorable situation, and avoid using this kind of information. Secondly, several features are hard to use in classification, because their values are not specific to any of the classes that we want to recognize. Their contribution to accuracy is therefore unclear, while they certainly add overhead to the recognition problem.

For the selection, we have used information from the literature, in particular from a recent study on the behavior of Machine-to-Machine (M2M) communication [43]. This was complemented by our own observation of the IoT flows. For instance, we notice that unlike traditional domestic traffic, the IoT flows tend to have a larger portion of upload packets, with a corresponding large proportion of ACK packets in the downlink. Therefore, the proportions of uplink and downlink packets could be considered as good proxies for our classification. Some of these attributes are not directly generated by the Tstat default distribution. The software was therefore modified to compute the following parameters:

- rate_ack_c/s = the number of ACK packets from server or client divided by the total number of packets from server or client. Measures the proportion of data versus the acknowledge of the data.
- rate_pkt_c/s = number of packets from server or client divided by the total number of packets. Measures the overall fraction of uplink and downlink traffic, over the total traffic.
- rate_bytes_c/s = number of bytes from server or client, divided by the total number of bytes. Similar to the previous, counts the bytes instead of the packets.

In addition to these, we have recently added the computation of the average inter arrival time, as the total flow duration divided by the total number of packets, and its standard deviation. These parameters have not yet been used for classification, and are part of our future work. Finally, because our captures do not always include the beginning of flows, we have modified the software to generate round trip time also for the incomplete flows.

The final attributes contained in the `.arff` file are therefore the following:

```
@attribute rate_ack_c:15 numeric
```

```
@attribute rate_bytes_c:16 numeric
@attribute rate_pkt_c:17 numeric
@attribute rate_ack_s:32 numeric
@attribute rate_bytes_s:33 numeric
@attribute rate_pkt_s:34 numeric
@attribute c_rtt_avg:51 numeric
@attribute c_rtt_min:52 numeric
@attribute c_rtt_max:53 numeric
@attribute c_rtt_std:54 numeric
@attribute s_rtt_avg:58 numeric
@attribute s_rtt_min:59 numeric
@attribute s_rtt_max:60 numeric
@attribute s_rtt_std:61 numeric
@attribute Protocol {IoT,NON_IoT}
```

Next, we will look at the classification methods employed and presents the results.

## 5.2 Clustering

The first method that we have used for classification is a semi-supervised clustering approach, based on the SeLeCT self learning classifier proposed by Grimaudo et al. [22]. We use a simpler procedure whereby we explore the optimal number of clusters required for a correct classification, focusing especially on the classification of the IoT flows.

We proceed as follows. Under the "Cluster" tab in Weka, we select the SimpleKMeans clustering algorithm. Using the defaults, we instruct the algorithm to ignore the IoT/NON_IoT label given to each flow, making the approach unsupervised. In other words, the algorithm will try to determine classes irrespective of the label that was assigned in the first place. Clustering is then run several times, progressively increasing the number of clusters. Ideally, two clusters would be sufficient, but naturally the unsupervised method is unable to aggregate IoT and non-IoT flows so that they are completely separated. With several clusters, instead, we might find smaller aggregates which are mostly IoT or mostly non-IoT. To make the determination for each cluster, in the Cluster mode in Weka we select the "Classes to clusters evaluation" option, using the IoT/NON_IoT label, so that each cluster is denoted with the number of actual IoT and NON_IoT flows that belong to the cluster. Clusters which have a majority of IoT flows are then labeled as IoT, while the others are labeled as NON_IoT. This is the supervised step of the approach: while clusters are identified based solely on the flow features, the destination of the cluster is determined based on the previous knowledge of the flow classification.

The first set of experiments makes use of the initial dataset, comprising mostly the *simulated IoT flows*. We expect clustering to work reasonably on this set, at least for the IoT traffic, since the virtual sensors are less diverse than actual sensors. The distribution of the flow parameters can be shown in Weka through the "Visualize" tab, which displays the position of each element of the dataset as a function of every pair of two attributes. Figure 5.1 shows an example of a two-dimensional plot of the acknowledge rate from the client (X axis) and from the server (Y axis), with IoT traffic in blue, and non-IoT traffic in red. One can clearly make out areas where the IoT traffic is more concentrated, which are good candidates for a cluster.

The results are presented in Table 5.3, which shows which of the clusters (its index) were labeled as IoT (contained a majority of IoT flows), with the corresponding number of actual IoT flows and NON_IoT flows that it contains. With 2 clusters, none of the clusters has a majority of IoT flows, so all flows are classified as NON_IoT. As the number of cluster increases, a few clusters appear with a majority of IoT flows, and are therefore labeled as IoT. For instance, with 50 clusters, 4 clusters are labeled as IoT (those with index 12, 29, 33 and 47). We observe that the number of IoT flows correctly categorized as IoT flows increases up to 50 clusters. Increasing the number of clusters gives no improvement, in fact the number slightly decreases. The number of NON_IoT flows incorrectly categorized as IoT, on the other hand, steadily decreases as the number of clusters increases. A division in 50 clusters seems to provide the best trade off. Figure 5.2 shows in dark color the four clusters labeled as IoT traffic for the 50-cluster case. Comparing with Figure 5.1, it is clear that most

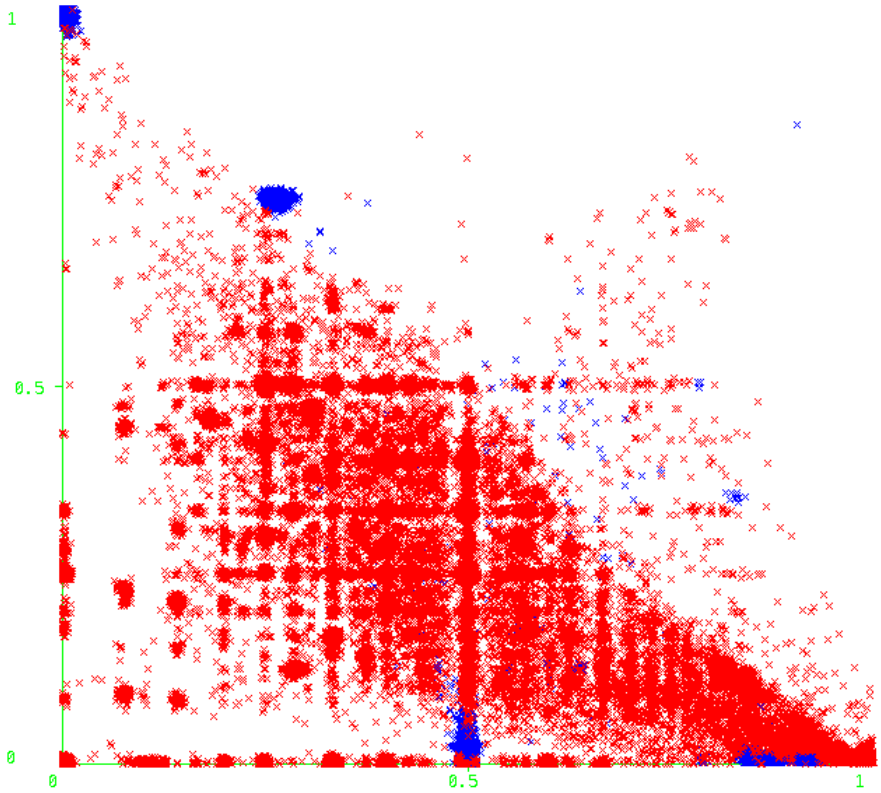



Figure 5.1: Acknowledge rate of the client (X asix) and of the server (Y axis). IoT traffic is in blue, non-IoT traffic in red. Does not include Australian dataset.

of the actual IoT traffic is included in the clusters. However, some flows are missed, while a few extra ones are included. Increasing the number of clusters does provide better granularity, however this may increase the computational complexity of classification. To determine the cluster membership, one has to compute the distance of a flow from the computed cluster centers. On the other hand, because we have a binary classification, there are opportunities to prune the decision algorithm, and avoid checking the distance to clusters once it is determined that a flow does not belong to any of the IoT clusters, which are few compared to the overall number. An alternative approach is to generate a decision tree using the labeling obtained from clustering (as opposed to the original labeling), to automatically select the best decision points in terms of the traffic attributes. The evaluation of this strategy is part of our future work.

Table 5.4 shows more in detail the results of clustering. The first column reports the number of clusters. The second and third columns report the confusion matrix: for each class (shown in the last column), the table shows the number of flows that were included in a cluster which was labeled as IoT or NON_IoT, respectively. The following four columns give a summary of the performance: we compute the True Positive (TP) and the False Positive (FP) rates, as well as the Precision and Recall measures for both IoT and NON_IoT flows. The third row for each experiment is a weighted average of the values. As discussed, with 2 clusters all flows are classified as NON_IoT. This is equivalent to the ZeroR method discussed above, and therefore corresponds to the baseline. As the number of cluster increases, we get a better Recall for the IoT flows, reaching a maximum of 96.6% for the division in 50 clusters. As we increase the number of clusters, the overall performance slightly increases, although we are less accurate on the IoT flow.

We have conducted the same analysis including the 15,000 flows from the Australian deployment. The expectation is that the results will be somewhat less satisfactory, because of the increased diversity of the devices in use. Figure 5.3 shows the same distribution of acknowledge rate from client (X axis) and server (Y axis) for this case. While there still are areas which are clearly identifiable, overall the distribution of IoT flows is much more dispersed. Indeed, for the 50-cluster experiment, we find 9 clusters which contain a majority of IoT flows for a total of only 13266 out of 22843 flows. The

Table 5.3: IoT clusters for different number of clusters.

| Clusters | Cluster index | IoT flows | NON_IoT flows |
|---|---|---|---|
| 2 | | | |
| | **Total** | **0** | **0** |
| 5 | 0 | 3006 | 863 |
| | **Total** | **3006** | **863** |
| 10 | 0 | 3006 | 454 |
| | 8 | 3107 | 680 |
| | **Total** | **6113** | **1134** |
| 20 | 12 | 3107 | 495 |
| | 13 | 3003 | 360 |
| | **Total** | **6110** | **855** |
| 30 | 4 | 1392 | 440 |
| | 12 | 3106 | 209 |
| | 29 | 2259 | 292 |
| | **Total** | **6757** | **941** |
| 50 | 12 | 3106 | 231 |
| | 29 | 2215 | 280 |
| | 33 | 1384 | 246 |
| | 47 | 791 | 147 |
| | **Total** | **7496** | **904** |
| 70 | 12 | 3106 | 159 |
| | 29 | 323 | 0 |
| | 47 | 748 | 134 |
| | 50 | 1935 | 287 |
| | 62 | 1377 | 205 |
| | **Total** | **7489** | **785** |
| 100 | 12 | 2811 | 142 |
| | 29 | 231 | 0 |
| | 50 | 223 | 0 |
| | 73 | 968 | 44 |
| | 79 | 295 | 0 |
| | 82 | 411 | 172 |
| | 83 | 709 | 136 |
| | 99 | 1833 | 287 |
| | **Total** | **7481** | **781** |

confusion matrix is therefore far from ideal, as shown by the following table which includes the accuracy details.

| Clusters | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| 50 | 13266 | 9577 | 58.1% | 5.4% | 77.9% | 58.1% | 58.6% | IoT |
| | 3767 | 65590 | 94.6% | 41.9% | 87.3% | 94.6% | 58.6% | NON_IoT |
| | | | 90.9% | 38.3% | 86.3% | 90.9% | 58.6% | Average |
| 100 | 15001 | 7842 | 65.7% | 3.4% | 86.4% | 65.7% | 68.8% | IoT |
| | 2353 | 67004 | 96.6% | 34.3% | 89.5% | 96.6% | 68.8% | NON_IoT |
| | | | 93.5% | 31.2% | 89.2% | 93.5% | 68.8% | Average |

The situation slightly improves when using 100 clusters, as shown above, however the precision is still fairly low, and the computational complexity of determining cluster membership increases. One of the reason why clustering does not provide good performance in this case is that the different

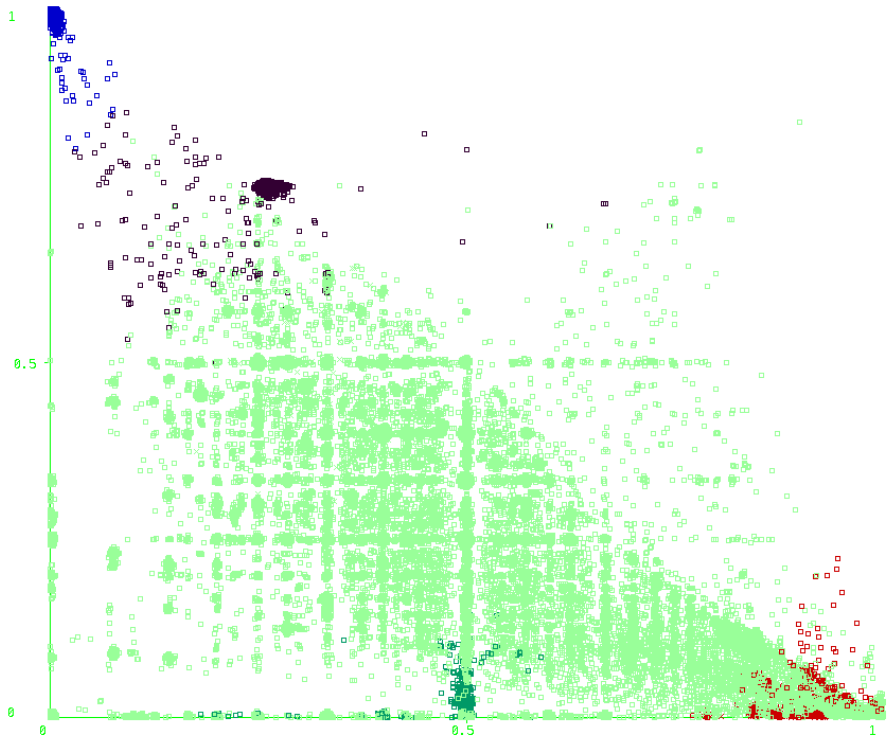



Figure 5.2: Acknowledge rate of the client (X asix) and of the server (Y axis). IoT labeled clusters shown in dark color, non-IoT clusters in light green. Does not include Australian dataset.

Table 5.4: Clustering results

| Clusters | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 7762 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | IoT |
| | 0 | 69357 | 1.0% | 1.0% | 89.9% | 100.0% | 0.0% | NON_IoT |
| | | | 89.9% | 89.9% | 80.9% | 89.9% | 0.0% | Average |
| 5 | 3006 | 4756 | 38.7% | 1.2% | 77.7% | 38.7% | 51.7% | IoT |
| | 863 | 68494 | 98.8% | 61.3% | 93.5% | 98.8% | 51.7% | NON_IoT |
| | | | 92.7% | 55.2% | 91.9% | 92.7% | 51.7% | Average |
| 10 | 6113 | 1649 | 78.8% | 1.6% | 84.4% | 78.8% | 79.5% | IoT |
| | 1134 | 68223 | 98.4% | 21.2% | 97.6% | 98.4% | 79.5% | NON_IoT |
| | | | 96.4% | 19.3% | 96.3% | 96.4% | 79.5% | Average |
| 20 | 6110 | 1652 | 78.7% | 1.2% | 87.7% | 78.7% | 81.3% | IoT |
| | 855 | 68502 | 98.8% | 21.3% | 97.6% | 98.8% | 81.3% | NON_IoT |
| | | | 96.7% | 19.3% | 96.6% | 96.7% | 81.3% | Average |
| 30 | 6757 | 1005 | 87.1% | 1.4% | 87.8% | 87.1% | 86.0% | IoT |
| | 941 | 68416 | 98.6% | 12.9% | 98.6% | 98.6% | 86.0% | NON_IoT |
| | | | 97.5% | 11.8% | 97.5% | 97.5% | 86.0% | Average |
| 50 | 7496 | 266 | 96.6% | 1.3% | 89.2% | 96.6% | 92.0% | IoT |
| | 904 | 68453 | 98.7% | 3.4% | 99.6% | 98.7% | 92.0% | NON_IoT |
| | | | 98.5% | 3.2% | 98.6% | 98.5% | 92.0% | Average |
| 70 | 7489 | 273 | 96.5% | 1.1% | 90.5% | 96.5% | 92.7% | IoT |
| | 785 | 68572 | 98.9% | 3.5% | 99.6% | 98.9% | 92.7% | NON_IoT |
| | | | 98.6% | 3.3% | 98.7% | 98.6% | 92.7% | Average |
| 100 | 7481 | 281 | 96.4% | 1.1% | 90.5% | 96.4% | 92.7% | IoT |
| | 781 | 68576 | 98.9% | 3.6% | 99.6% | 98.9% | 92.7% | NON_IoT |
| | | | 98.6% | 3.4% | 98.7% | 98.6% | 92.7% | Average |

features contribute symmetrically to the Euclidean distance from the cluster centroid. This is less of a concern with more homogeneous features, but induces confusion when traffic has a higher degree of

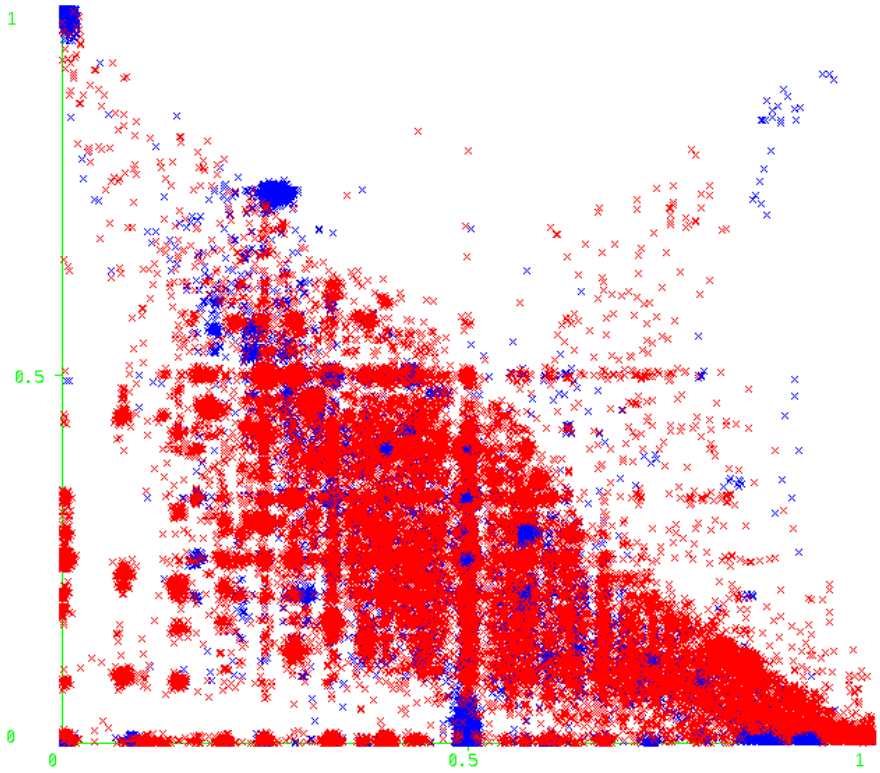



Figure 5.3: Acknowledge rate of the client (X asix) and of the server (Y axis). IoT traffic is in blue, non-IoT traffic in red. Includes Australian dataset.

overlapping. In the next section we explore other classification methods, which can selectively favor different features to achieve better performance.

## 5.3 Classification

Our second approach is to consider supervised learning algorithms, trained with our dataset. We have used several methods, which are provided by Weka, and computed their performance using 10-fold cross validation. We consider each method in the following subsections. The basic idea is to use decision procedures that can represent nonlinear boundaries across the feature space.

### 5.3.1 Multilevel Perceptron

A *Multilevel Perceptron* is a neural network that contains one or more internal layers. We have constructed and evaluated three different neural network configurations, varying the number of neurons in the hidden layer:

- NN10: one hidden layer with 10 neurons;
- NN20: one hidden layer with 20 neurons;
- NN10.10: two hidden layers with 10 neurons each;

All networks have two output neurons that provide the classification result. The results are summarized in Table 5.5, where we again consider the simulated and the complete dataset separately.

The results show that the neural network classifier behaves better than clustering. The accuracy is rather high for the dataset with the simulated traffic, while it is less satisfactory for the complete dataset with real traffic, although still superior to clustering. The adoption of a wider hidden layer provides some limited improvement, while going from one to two hidden layers (with a constant number of neurons) does not change the results appreciably. A deep neural network may be able to extract more significant features from the attributes.

Table 5.5: Results of the Neural Network classifier

| Config | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| NN10 | 7550 | 212 | 97.3% | 0.1% | 99.4% | 97.3% | 98.1% | IoT |
| Simulated | 49 | 69308 | 99.9% | 2.7% | 99.7% | 99.9% | 98.1% | NON_IoT |
| | | | 99.7% | 2.5% | 99.7% | 99.7% | 98.1% | Average |
| NN20 | 7578 | 184 | 97.6% | 0.1% | 99.3% | 97.6% | 98.3% | IoT |
| Simulated | 52 | 69305 | 99.9% | 2.4% | 99.7% | 99.9% | 98.3% | NON_IoT |
| | | | 99.7% | 2.1% | 99.7% | 99.7% | 98.3% | Average |
| NN10.10 | 7573 | 189 | 97.6% | 0.1% | 99.1% | 97.6% | 98.2% | IoT |
| Simulated | 68 | 69289 | 99.9% | 2.4% | 99.7% | 99.9% | 98.2% | NON_IoT |
| | | | 99.7% | 2.2% | 99.7% | 99.7% | 98.2% | Average |
| NN10 | 17940 | 4903 | 78.5% | 2.0% | 92.9% | 78.5% | 81.2% | IoT |
| Complete | 1381 | 67976 | 98.0% | 21.5% | 93.3% | 98.0% | 81.2% | NON_IoT |
| | | | 93.2% | 16.6% | 93.2% | 93.0% | 81.2% | Average |
| NN20 | 18677 | 4166 | 81.8% | 1.7% | 94.1% | 81.8% | 84.1% | IoT |
| Complete | 1169 | 68188 | 98.3% | 18.2% | 94.2% | 98.3% | 84.1% | NON_IoT |
| | | | 94.2% | 14.1% | 94.2% | 94.2% | 84.1% | Average |
| NN10.10 | 18697 | 4146 | 81.9% | 2.6% | 91.2% | 81.9% | 82.3% | IoT |
| Complete | 1801 | 67556 | 97.4% | 18.1% | 94.2% | 97.4% | 82.3% | NON_IoT |
| | | | 93.5% | 14.3% | 93.5% | 93.5% | 82.3% | Average |
| NN30 | 18751 | 4092 | 82.1% | 1.4% | 94.9% | 82.1% | 84.8% | IoT |
| Complete | 1005 | 68352 | 98.6% | 17.9% | 94.4% | 98.6% | 84.8% | NON_IoT |
| | | | 94.5% | 13.8% | 94.5% | 94.5% | 84.8% | Average |

### 5.3.2 Support Vector Machines

In a simple linear classifier, the boundaries between classes are straight lines, which are too simple for our application. A Support Vector Machine (SVM) uses linear models, however it transform the input space through a nonlinear mapping. Hence, the linear model in the new space is no longer linear when mapped back in the original space. The results achieved by this classifier, shown in Table 5.6,

Table 5.6: Results of the Support Vector Machine classifier

| Config | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| SVM | 3894 | 3868 | 50.2% | 0.1% | 97.4% | 50.2% | 67.9% | IoT |
| Simulated | 104 | 69253 | 99.9% | 49.8% | 94.7% | 99.9% | 67.9% | NON_IoT |
| | | | 94.8% | 44.8% | 95.0% | 94.8% | 67.9% | Average |

are not particularly encouraging. It has been implemented with the default Weka parameters for the simulated dataset only.

### 5.3.3 Naïve Bayes

The Naïve Bayes method is based on Bayes's rule and "naïvely" assumes independence of the attributes. The assumption that attributes are independent is simplistic, however the obtained classifier is simple to implement, and often works well when tested on actual datasets. This is not unfortunately the case for our traffic data. Even with only the simulated flows, while IoT flows are mostly correctly classified, traditional traffic is often misinterpreted as IoT traffic. The results are shown below:

| Config | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| Naïve Bayes | 7651 | 111 | 98.6% | 21.2% | 34.2% | 98.6% | 51.3% | IoT |
| Simulated | 14703 | 54654 | 78.8% | 1.4% | 99.8% | 78.8% | 51.3% | NON_IoT |
| | | | 80.8% | 3.4% | 93.2% | 80.8% | 51.3% | Average |

### 5.3.4 J48 Classification Tree

Classification trees operate by selecting a path through a tree, branching at every node by comparing the value of one of the parameters against a particular threshold. A result is obtained upon reaching a leaf of the tree. The branching variable and the threshold at each node are the parameters learned during training. In particular, we use the J48 algorithm, a Java implementation of the C4.5 algorithm [25].

Classification trees have been shown to perform well in protocol recognition [21]. One of the advantages of this approach is that recognition does not require complex mathematical operations, other than the comparison of parameters with some constant value. Thus, from this point of view, they are particularly well suited to the implementation on network processors, which are often not optimized for math algorithms. In addition, traversing the tree is typically fast, an important feature when operating at line speed.

We have generated several classification tree, for the initial simulated dataset and for the complete dataset. We have also analyzed the influence of the different parameters on both performance and tree size. As usual, the accuracy is evaluated through the resulting confusion matrix using 10-fold cross validation. Tree size, on the other hand, can be evaluated in terms of number of nodes and tree depth.

Our first experiment deals with the simulated and the complete dataset using the full set of attributes. The trees performs particularly well, as shown in Table 5.7 where the confusion matrix highlights that only a few of the flows are missclassified. In particular, the performance is superior

Table 5.7: Accuracy of the classification trees with all attributes.

| Config | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| J48 | 7686 | 76 | 99.0% | 0.1% | 99.4% | 99.0% | 99.1% | IoT |
| Simulated | 47 | 69310 | 99.9% | 1.0% | 99.9% | 99.9% | 99.1% | NON_IoT |
| | | | 99.8% | 0.9% | 99.8% | 99.8% | 99.1% | Average |
| J48 | 22449 | 394 | 98.3% | 0.5% | 98.5% | 98.3% | 97.8% | IoT |
| Complete | 345 | 69012 | 99.5% | 1.7% | 99.4% | 99.5% | 97.8% | NON_IoT |
| | | | 99.2% | 1.4% | 99.2% | 99.2% | 97.8% | Average |

to any of the methods that we have analyzed so far, with an average precision and recall that exceed 99% for both datasets. By way of example, Figure 5.4 shows the tree generated for the simulated case. The green leaves represent the NON_IoT classification, while the yellow boxes the IoT. The internal nodes indicate the test variable and the condition, with the branches recording the outcome of the condition. The tree can also be shown in textual form, as shown below (beginning of the tree only), which is easier to read for analysis.

```
J48 pruned tree
------------------

rate_ack_s:32 <= 0.730766
|   s_rtt_min:59 <= 0.026
|   |   c_rtt_std:54 <= 1.685583
|   |   |   rate_ack_c:15 <= 0.482757: NON_IoT (2419.0)
|   |   |   rate_ack_c:15 > 0.482757
|   |   |   |   rate_pkt_c:17 <= 0.547618
|   |   |   |   |   rate_pkt_c:17 <= 0.49624: NON_IoT (250.0)
|   |   |   |   |   rate_pkt_c:17 > 0.49624
|   |   |   |   |   |   rate_ack_c:15 <= 0.629627: NON_IoT (211.0/2.0)
|   |   |   |   |   |   rate_ack_c:15 > 0.629627: IoT (16.0/1.0)
|   |   |   |   rate_pkt_c:17 > 0.547618
|   |   |   |   |   rate_bytes_c:16 <= 0.03462: NON_IoT (16.0)
|   |   |   |   |   rate_bytes_c:16 > 0.03462
|   |   |   |   |   |   s_rtt_avg:58 <= 7.940603
|   |   |   |   |   |   |   rate_bytes_c:16 <= 0.655367: IoT (55.0/1.0)
|   |   |   |   |   |   |   rate_bytes_c:16 > 0.655367: NON_IoT (3.0/1.0)
|   |   |   |   |   |   s_rtt_avg:58 > 7.940603
|   |   |   |   |   |   |   rate_ack_s:32 <= 0.190475: NON_IoT (11.0)
```

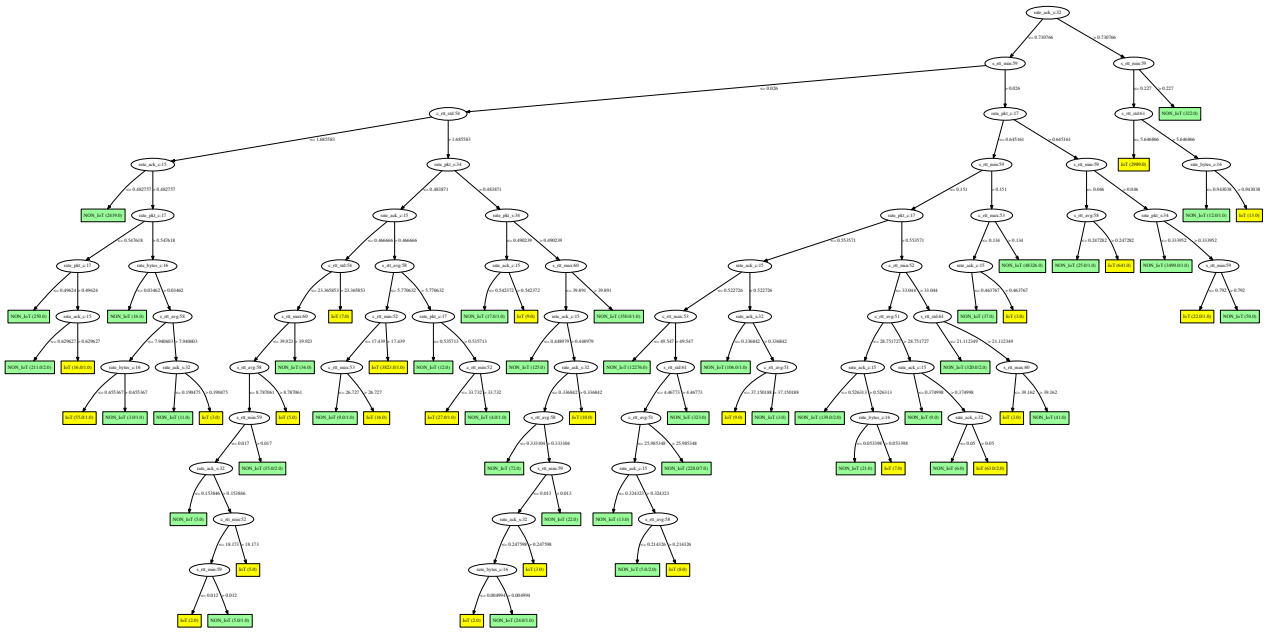


Figure 5.4: The generated J48 classification tree for the simulated dataset.

```
|   |   |   |   |   |   |   rate_ack_s:32 > 0.190475: IoT (3.0)
|   |   c_rtt_std:54 > 1.685583
|   |   |   rate_pkt_s:34 <= 0.483871
|   |   |   |   rate_ack_c:15 <= 0.466666
|   |   |   |   |   c_rtt_std:54 <= 23.365853
. . .
```

Notice how the leaves include not only the classification label, but also (in parenthesis) the number of flows of the class (and, if present, of the other class) in the training set that lead to the leaf. This number is shown also in each of the leaves on Figure 5.4 (although the size of the figure makes it barely readable).

Table 5.8 shows the tree information in terms of computational complexity. The first column

Table 5.8: Complexity of the classification trees with all attributes.

| Config | Size | Leaves | Min depth | Max depth | Avg depth |
|---|---|---|---|---|---|
| J48 Simulated | 125 | 63 | 1 | 11 | 4.4 |
| J48 Complete | 635 | 318 | 2 | 23 | 7.4 |

reports the total size of the tree (number of nodes), while the second column counts the number of leaves in the tree. The size of the tree gives an estimate of the amount of memory required to store the tree information. The following three columns provide information regarding the depth of the tree: the minimum and the maximum depth to reach a leaf, as well as the average, where the depth is weighted by the number of flows in the training set that are associated with each particular leaf. The data shows that the lower variability associated with the simulated flows results in a much smaller and shallower tree for classification.

It is interesting to study the influence of each attribute on the classification accuracy. This could be useful, for instance, to select only a subset of the attributes that provide most of the performance. In fact, while a smaller subset does not necessarily imply a smaller tree, however it does require fewer resources to compute the statistics during operation.

To select the most relevant attributes, we proceed in two ways. The first is a greedy search, whereby we evaluate the classification performance using progressively more attributes. Hence, we

start by evaluating the performance of all trees that use only one attribute, and keep the attribute that provides the best performance (we rank the trees on the basis of their MCC). Then, we evaluate all trees with two attributes, having fixed the first in the previous step. We continue this way until we see that the performance is sufficiently high. The results of this search for both the simulated and complete datasets are shown in Table 5.9. The results show that performance increases quickly with

Table 5.9: Greedy search for best attribute.

| Config | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| J48 | 7411 | 351 | 95.5% | 0.7% | 93.8% | 95.5% | 94.0% | IoT |
| Simulated | 487 | 68870 | 99.3% | 4.5% | 99.5% | 99.3% | 94.0% | NON_IoT |
| Attr.: 8 | | | 98.9% | 4.1% | 98.9% | 98.9% | 94.0% | Average |
| J48 | 7598 | 164 | 97.9% | 0.1% | 98.9% | 97.9% | 98.2% | IoT |
| Simulated | 84 | 69273 | 99.9% | 2.1% | 99.8% | 99.9% | 98.2% | NON_IoT |
| Attr.: 8,2 | | | 99.7% | 1.9% | 99.7% | 99.7% | 98.2% | Average |
| J48 | 7657 | 105 | 98.6% | 0.1% | 99.4% | 98.6% | 98.9% | IoT |
| Simulated | 49 | 69308 | 99.9% | 1.4% | 99.8% | 99.9% | 98.9% | NON_IoT |
| Attr.: 8,2,1 | | | 99.8% | 1.2% | 99.8% | 99.8% | 98.9% | Average |
| J48 | 7672 | 90 | 98.8% | 0.1% | 99.5% | 98.8% | 99.1% | IoT |
| Simulated | 42 | 69315 | 99.9% | 1.2% | 99.9% | 99.9% | 99.1% | NON_IoT |
| Attr.: 8,2,1,3 | | | 99.8% | 1.0% | 99.8% | 99.8% | 99.1% | Average |
| J48 | 19290 | 3553 | 84.4% | 6.5% | 81.2% | 84.4% | 77.0% | IoT |
| Complete | 4475 | 64882 | 93.5% | 15.6% | 94.8% | 93.5% | 77.0% | NON_IoT |
| Attr.: 8 | | | 91.3% | 13.3% | 91.4% | 91.3% | 77.0% | Average |
| J48 | 21950 | 893 | 96.1% | 1.3% | 96.2% | 96.1% | 94.9% | IoT |
| Complete | 868 | 68489 | 98.7% | 3.9% | 98.7% | 98.7% | 94.9% | NON_IoT |
| Attr.: 8,12 | | | 98.1% | 3.3% | 98.1% | 98.1% | 94.9% | Average |
| J48 | 22334 | 509 | 97.8% | 0.6% | 98.1% | 97.8% | 97.3% | IoT |
| Complete | 430 | 68927 | 99.4% | 2.2% | 99.3% | 99.4% | 97.3% | NON_IoT |
| Attr.: 8,12,2 | | | 99.0% | 1.8% | 99.0% | 99.0% | 97.3% | Average |
| J48 | 22393 | 450 | 98.0% | 0.5% | 98.4% | 98.0% | 97.6% | IoT |
| Complete | 370 | 68987 | 99.5% | 2.0% | 99.4% | 99.5% | 97.6% | NON_IoT |
| Attr.: 8,12,2,1 | | | 99.1% | 1.6% | 99.1% | 99.1% | 97.6% | Average |

the addition of more attributes. In both the simulated and the complete dataset, attributes 1, 2 and 8 are selected among the first four. These correspond to the fraction of acknowledge from the client to the server (1), the fraction of bytes from client to server (2), and the client minimum round trip time (8). In the simulated case, attribute number 3, the fraction of packets from client to server, completes the set, whereas for the complete case attribute 12, minimum server round trip time, is used. The size of the trees corresponding to the different choices of attributes is shown in Table 5.10.

Table 5.10: Complexity of the classification trees with selected attributes.

| Config | Size | Leaves |
|---|---|---|
| J48 Simulated, Attr.: 8 | 27 | 14 |
| J48 Simulated, Attr.: 8,2 | 111 | 56 |
| J48 Simulated, Attr.: 8,2,1 | 113 | 57 |
| J48 Simulated, Attr.: 8,2,1,3 | 137 | 69 |
| J48 Complete, Attr.: 8 | 99 | 50 |
| J48 Complete, Attr.: 8,12 | 311 | 156 |
| J48 Complete, Attr.: 8,12,2 | 533 | 267 |
| J48 Complete, Attr.: 8,12,2,1 | 571 | 286 |

The second mechanism for attribute selection makes use of the facility provided by the Weka framework. The "Select attributes" tab can be used to explore the space of attributes using different strategies. We perform a Wrapper Subset Evaluation, which is a scheme similar to the one employed above, using a Greedy Step-wise incremental search. In all cases, we select the J48 algorithm for evaluation. The results for the Simulated and the complete dataset are shown below:

```
Simulated dataset
Selected attributes: 1,2,3,6,7,8,11,13 : 8
                     rate_ack_c:15
                     rate_bytes_c:16
                     rate_pkt_c:17
                     rate_pkt_s:34
                     c_rtt_avg:51
                     c_rtt_min:52
                     s_rtt_avg:58
                     s_rtt_max:60

Complete dataset
Selected attributes: 1,2,3,4,5,8,9,12 : 8
                     rate_ack_c:15
                     rate_bytes_c:16
                     rate_pkt_c:17
                     rate_ack_s:32
                     rate_bytes_s:33
                     c_rtt_min:52
                     c_rtt_max:53
                     s_rtt_min:59
```

In both cases, Weka selects eight attributes out of the available 14, including the ones that we have determined using the manual procedure above. The results of generating the classification tree, shown in Table 5.11, are somewhat surprising. The accuracy is in fact slightly better than that of the tree that

Table 5.11: Classification tree performance, Weka selected attributes.

| Config | IoT | NON_IoT | TP rate | FP rate | Precision | Recall | MCC | Class |
|---|---|---|---|---|---|---|---|---|
| J48 | 7701 | 61 | 99.2% | 0.1% | 99.5% | 99.2% | 99.3% | IoT |
| Simulated | 36 | 69321 | 99.9% | 0.8% | 99.9% | 99.9% | 99.3% | NON_IoT |
| | | | 99.9% | 0.7% | 99.9% | 99.9% | 99.3% | Average |
| J48 | 22475 | 368 | 98.4% | 0.5% | 98.6% | 98.4% | 98.0% | IoT |
| Complete | 315 | 69042 | 99.5% | 1.6% | 99.5% | 99.5% | 98.0% | NON_IoT |
| | | | 99.3% | 1.3% | 99.3% | 99.3% | 98.0% | Average |

uses all the attributes together. This may be an indication that there is some degree of "overfitting", i.e., that there are too many parameters to choose from.

## 5.4 Final Considerations

We have shown that semi-supervised clustering works reasonably well in the case of homogeneous traffic. However, for the real world deployment clustering does not achieve satisfying performance. We have then considered supervised methods, such as neural networks and classification trees. The results of 10-fold cross validation show that classification trees provide the best results, with performance in excess of 99% accuracy. Attribute selection is used to is used to narrow down the set of attributes and to avoid overfitting. This was the first part of our work, in the next section we will explore the

analysis of the packets frequency.

# 6 Preliminaries experiments on temporal series construction

We start with the analysis of previous studies about the use of Fast Fourier Transform to detect internet flows [17, 23, 29, 53]. In the first place, we have investigated to find the best features to use to compute the Fast Fourier Transform. At first, we tried to extract for each flow, the number of packets sent and received in a unit of time (sampling period). The main problem here was to compare flows with different sampling periods, beacuse the different granularity gives not good results in classifications with decision trees. Furthermore, we tried to use the same sampling period for each flow, but the results have not so much improved. Secondly, we attempted to change the classification method, trying to compute the euclidean distance between points of different FFT series. If the euclidean distance was less than epsilon (threshold set by doing tests), then the two FFT series are similar, otherwise not. The results with this classification method are discreet, the problem here is set a balanced epsilon threshold to have not so much false positive, in this case many IoT flows classified non-IoT or viceversa. At the end, we tried to construct series using the length and the inter-arrival-time between packets. For this process, the classification with decision trees performs very well, using both 10-fold cross validation or using testset from different environment. We analyse this method in chapter 7. In the next sections, first we see an overview of Fast Fourier Transform, the datasets used, a data visualization, and then, we analyse in depth the packets rate experiments and we'll evaluate the perfomance of them.

## 6.1 Fast Fourier Transform

The fast fourier transform is an optimized algorithm to compute the discrete fourier transform. The DFT (discrete fourier transform) is defined as:

$$X_q = \sum_{k=0}^{N-1} x_k e^{-j\frac{2\pi}{N}kq} \qquad q = 0, 1, ..., N-1 \tag{6.1}$$

Where $X_q$ is the frequency value at position q, $x_k$ is the time series at the k-th timestamp.

### 6.1.1 Event Based Sampling

The sample instant is sampled every time an event occurs [39]. An event is when the amplitude of the signal passes a pre-defined level. The classical sampling technique measures the amplitude of a continuous time signal y(t) at regular time intervals $T_s$.

$$y_k = y(kT_s), \qquad k = 1, 2...N \tag{6.2}$$

In event based sampling the signal is sampled every time the amplitude of the signal passes certain pre-defined levels

$$y(t_k) = y_k, \qquad k = 1, 2...N \tag{6.3}$$

where $y_k$ is the pre-defined amplitudes recorded at time instants $t_k$. It is also possible to consider the event domain signal as a time domain signal with varying sample period.

## 6.2 Packet Rate Overview

We show different techniques to compute the input time series. We use two methods, the first, is set a fixed sampling period $T_s$, the second was to set a variable sampling period $T_s$, and see how many

packets are sent in this interval for both experiments.

### 6.2.1 Dataset preparation

For these preliminary experiments we used a small subset of australian dataset [48, 49, 47]. The total of IoT flows used in this experiment is 520. Regarding the NON_IoT flows, we use a `.pcap` collected by me in a domestic environment. From them we extract a total of 320 NON_IoT flows. For some of those tests we used a small set of IoT simulated flows collected with MIMIC simulator [5].

### 6.2.2 Data visualization

As described previously one of the problem for our tests was to choose the sampling period. We tried to do different proofs using a different sampling period for every test. First we tried to set the sampling period to 100 microseconds, then to 200 microseconds and finally to 500 microseconds and 1 millisecond respectively. Now we will visualize a flow of "Amazon Echo", an IoT device from australian dataset [48, 47, 49], an IoT flow collected with MIMIC simulator [5], and a NON_IoT flow collected in browsing session in a domestic environment, sampled with different sampling period.

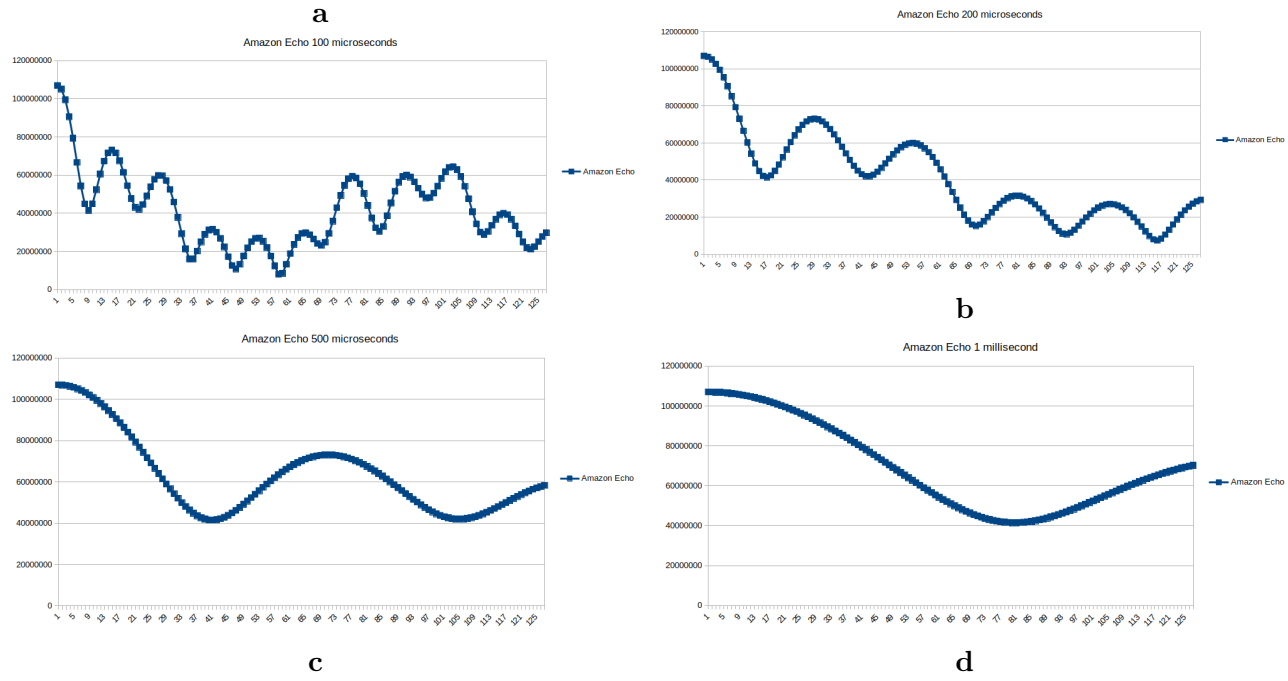



Figure 6.1: Amazon Echo 100 microseconds: (**a**) Amazon Echo 200 microseconds: (**b**) Amazon Echo 500 microseconds: (**c**) Amazon Echo 1 millisecond: (**d**)

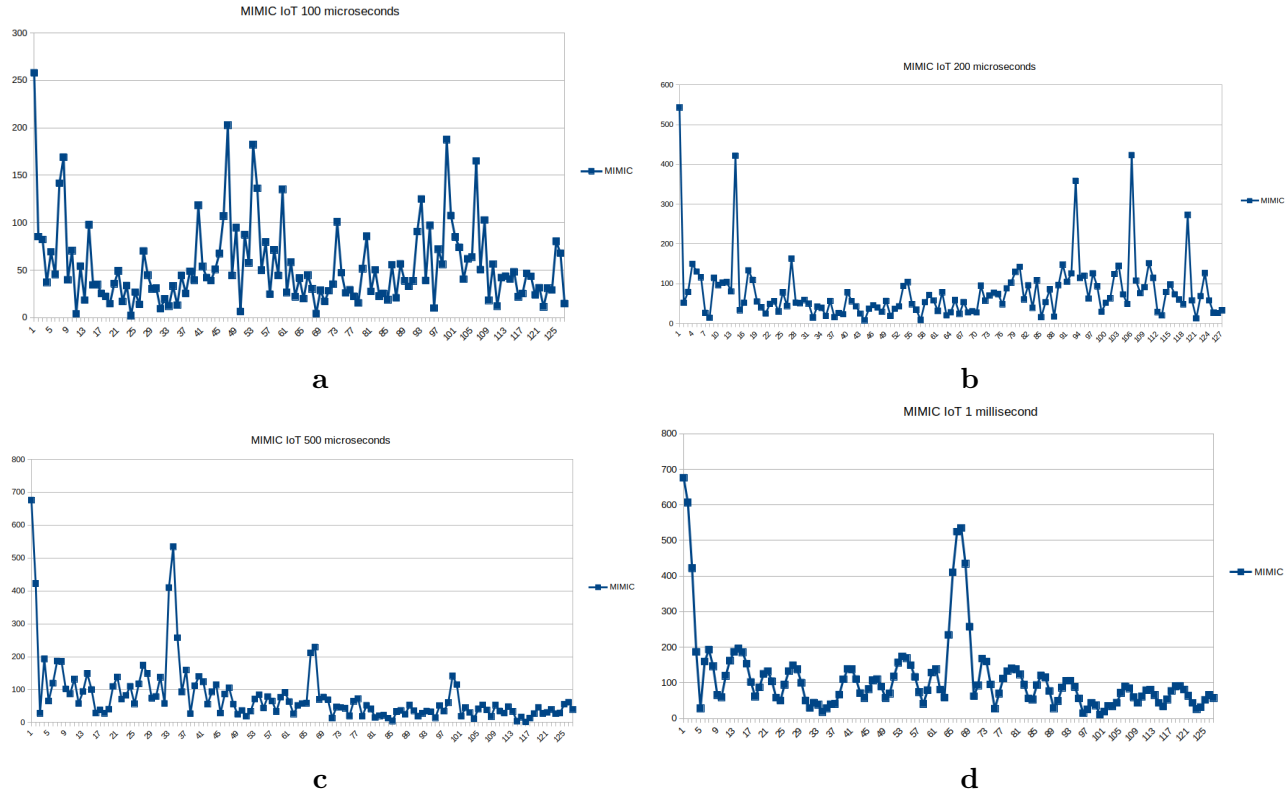



Figure 6.2: MIMIC 100 microseconds: (**a**) MIMIC 200 microseconds: (**b**) MIMIC 500 microseconds: (**c**) MIMIC 1 millisecond: (**d**)

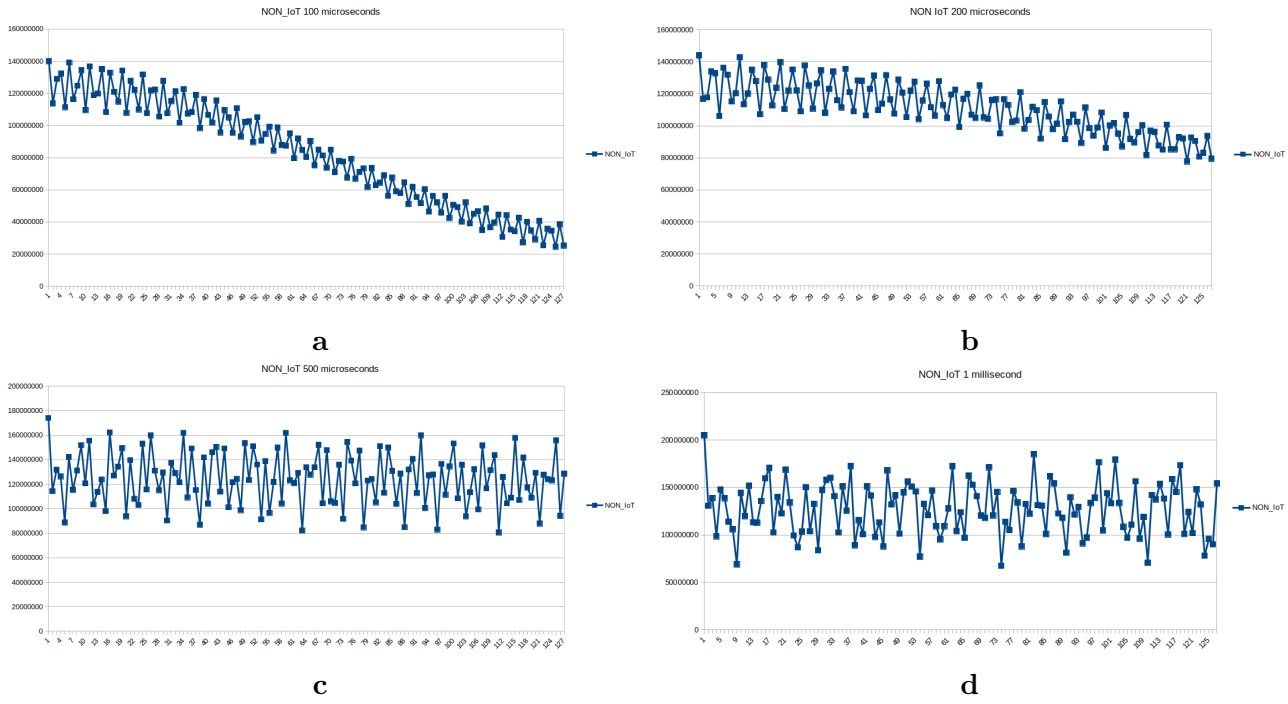



Figure 6.3: NON_IoT 100 microseconds: (**a**) NON_IoT 200 microseconds: (**b**) NON_IoT 500 microseconds: (**c**) NON_IoT 1 millisecond: (**d**)

We see in Figures 6.1 6.2 6.3 the different signals of the three different devices sampled with different sampling periods, 100 microseconds, 200 microseconds, 500 microseconds and 1 millisecond respectively. We can denote the differences between the sampling periods and the different granularities. With smaller sampling periods we see more signal details, but for longer communications and

when we have a long inter-arrival time between two packets we have to sample many times and most of them gives us not informations. For istance if the inter-arrival time between two packets is 10 milliseconds, and the sampling period is 100 microseconds, we have 100 samplings without packets. If the sampling period is too short we risk to have patterns like 0,0,0,...0. For this cases we need to increase the sampling period but, having a sampling period too high, carries us to lose too many details of the flow. For these reasons we need to find a good balance. Therefore to have a good balance we tried to use this sampling period [29]:

$$T = 0.1(\frac{D}{n}) \tag{6.4}$$

Where D is the duration of a connection, and n is the number of packets. The problem with this criteria was to compare different signal with different granularity, indeed a variable sampling period show us not good perfomance when we perform decision trees. We applied this criteria to the previous flows showed in Figures 6.1 6.2 6.3. The IoT flow from australian dataset was sampled with a sampling period of 555 microseconds, the IoT flow from MIMIC was sampled at 494 microseconds, and the NON_IoT flow was sampled with a sampling period of 31 milliseconds. In the Figure 6.4 we illustrate the different signals.

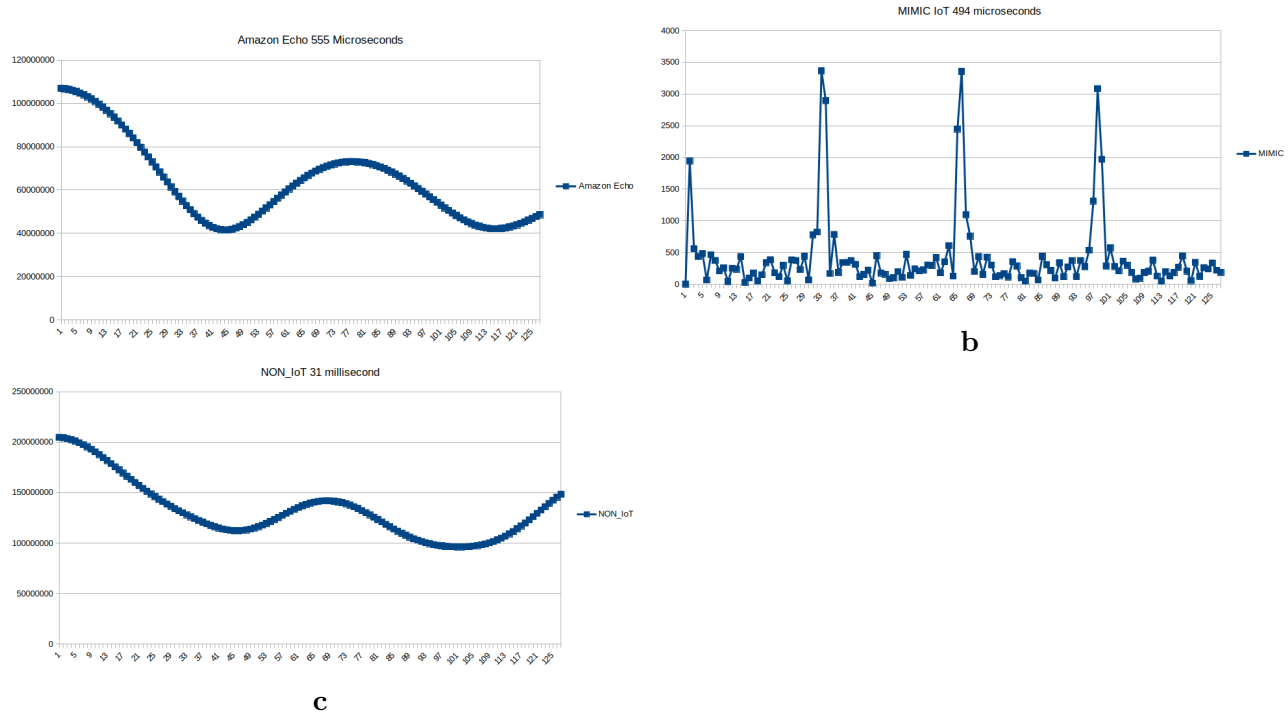
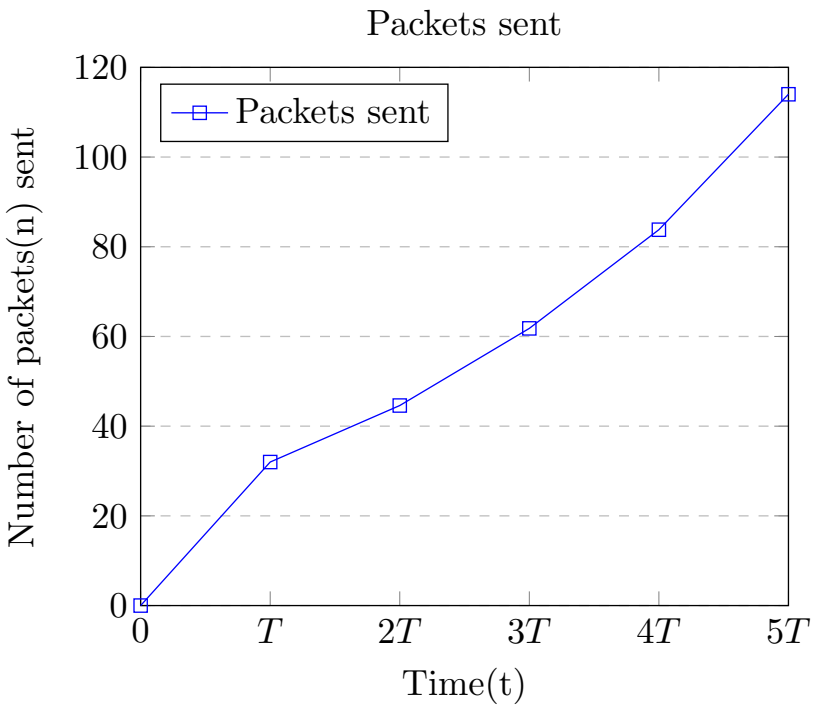



Figure 6.4: Amazon IoT 555 microseconds: (**a**) MIMIC IoT 494 microseconds: (**b**) NON_IoT 31 milliseconds: (**c**)

### 6.2.3 Packet Rate Procedure

As described in the previous section, the main problem was to choose the sampling period, because, if it was too short and the time between two packets was too long, the number of bins were insufficient to contain all the points. We set to 65536 the number of bins. Contrarily, if the sampling period was too long, we loose many details of the signal, because a coarse granularity gives us not good performance. If 65536 points were insufficient we used a divide et impera approach, fragmenting the signal in more FFT.

Now we illustrate a plot that shows in detail the sampling and how the time series $x_k$ is composed.




T represents the sampling period in time, instead on the y-axis we have the number of packets sent.

We set:

$$\Delta_k = x(kT) - x((k-1)T), \qquad k = 1, 2...N \tag{6.5}$$

Then we compute $x_k$ as:

$$x_k = \Delta_k \tag{6.6}$$

First of all, we have recorded 65536 points for every flows. Subsequently we have isolated N peaks, first 50 peaks and then less, to try different configurations. The peaks are the points with the maximum energy, and the highlights with more informations. Furthermore, the points before and after the peak has not considerated. The peaks are used as input for training the random forest decision tree.

### 6.2.4 Packet Rate Results

The training set is composed of 852 flows, 520 are IoT while 332 are non_IoT. For this experiment first we used respectively 50,30,10 peaks points taken by all the 65536 points of the spectrum. As algorithm we used random forest provided by the weka tool and for evaluation 10-fold cross validation. We tried different methods such as neural networks, but they performs worse than decision trees. The results are discreet, with 50 peaks the correctly classified istances are 774 (90.84%) while the incorrectly classified istance are 78 (9.15%) as shown in table 6.1.

Table 6.1: Results with 50 peaks

| Class | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 495 | 25 | 95.2% | 16.0% | 90.3% | 95.2% | 80.7% |
| NON_IoT | 279 | 54 | 84.0% | 4.8% | 91.8% | 84.0% | 80.7% |

Table 6.2: Results with 50 peaks

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 774 - 90.85% | 78 - 9.15% | 852 - 100% |

The confusion matrix is:

Table 6.3: Confusion matrix with 50 peaks

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 279 | 53 | NON_IoT |
| 25 | 495 | IoT |

The results with 30 peaks are shown in the table 6.4. Comparing the results we see that there are almost similar.

Table 6.4: Results with 30 peaks

| **Class** | **Correct** | **Wrong** | **TP rate** | **FP rate** | **Precision** | **Recall** | **MCC** |
|---|---|---|---|---|---|---|---|
| IoT | 491 | 29 | 94.4% | 14.5% | 91.1% | 94.4% | 80.9% |
| NON_IoT | 284 | 48 | 85.5% | 5.6% | 90.7% | 85.5% | 80.9% |

Table 6.5: Results with 30 peaks

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 775 - 90.96% | 77 - 9.04% | 852 - 100% |

The confusion matrix is:

Table 6.6: Confusion matrix with 30 peaks

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 284 | 48 | NON_IoT |
| 29 | 491 | IoT |

Finally we tried with 10 peaks as shown in table 6.7.

Table 6.7: Results with 10 peaks

| **Class** | **Correct** | **Wrong** | **TP rate** | **FP rate** | **Precision** | **Recall** | **MCC** |
|---|---|---|---|---|---|---|---|
| IoT | 491 | 29 | 94.4% | 10.8% | 93.2% | 94.4% | 83.9% |
| NON_IoT | 296 | 36 | 89.2% | 5.6% | 91.1% | 89.2% | 83.9% |

Table 6.8: Results with 10 peaks

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 787 - 92.37% | 65 - 7.63% | 852 - 100% |

The confusion matrix is:

Table 6.9: Confusion matrix with 10 peaks

| NON_IoT | IoT | Classified as |
|---|---|---|
| 296 | 36 | NON_IoT |
| 29 | 491 | IoT |

We see that with 10 peaks the results improve, more non_IoT flows are correctly classified compared with 30 and 50 peaks.

### 6.2.5 Clustering

Moreover we tried to perform clustering using simple-k means algorithm. We instruct the algorithm to ignore the IoT/NON_IoT label given to each flow, making the approach unsupervised. The algorithm will try to determine classes irrespective of the label that was assigned in the first place. Using 50 peaks as input we construct 20 cluster. Ideally, two clusters would be sufficient, but naturally the unsupervised method is unable to aggregate IoT and non-IoT flows so that they are completely separated. For every cluster we calculate how many IoT flows and how many non_IoT flows are collected in the same cluster. If IoT flows are more than non_IoT, we classify the cluster as IoT otherwise we classify them as non_IoT.

The results are presented in Table 6.10, which shows for every cluster how the flows are distributed.

Table 6.10: Flow Distribution

| Cluster index | IoT flows | NON_IoT flows | Total |
|---|---|---|---|
| 0 | 1 | 1 | 2 |
| 1 | 9 | 42 | 51 |
| 2 | 1 | 1 | 2 |
| 3 | 12 | 28 | 40 |
| 4 | 3 | 18 | 21 |
| 5 | 3 | 21 | 24 |
| 6 | 2 | 11 | 13 |
| 7 | 14 | 7 | 21 |
| 8 | 8 | 1 | 9 |
| 9 | 28 | 4 | 32 |
| 10 | 1 | 0 | 1 |
| 11 | 2 | 1 | 3 |
| 12 | 262 | 25 | 287 |
| 13 | 3 | 0 | 3 |
| 14 | 43 | 22 | 21 |
| 15 | 41 | 120 | 161 |
| 16 | 44 | 18 | 62 |
| 17 | 20 | 9 | 29 |
| 18 | 4 | 1 | 5 |
| 19 | 40 | 3 | 43 |

We denote that without labels, clustering approach is not able to distinguish, using the 50 peaks of the FFT, the IoT flows from NON_IoT. The cluster with index 12 contains a majority of IoT flows so we can label as IoT. On the other hand, an example of a cluster labeled as NON_IoT is the number 15 when we have a majority of NON_IoT flows. The mean of IoT flows per cluster is 59.1%.

### 6.2.6 Multilevel Perceptron

We have constructed and evaluated a neural network configuration, with one hidden layer with 26 neurons. All networks have two output neurons that provide the classification result. The results are

summarized in Table 6.11.

Table 6.11: Results of the Neural Network classifier

| Class | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 446 | 74 | 85.8% | 72.6% | 64.9% | 95.8% | 16.3% |
| NON_IoT | 91 | 241 | 27.4% | 14.2% | 55.2% | 27.4% | 16.3% |

Table 6.12: Results of the Neural Network classifier

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 537 - 63.02% | 315 - 36.98% | 852 - 100% |

The confusion matrix is:

Table 6.13: Results of the Neural Network classifier

| IoT | NON_IoT | Class |
|---|---|---|
| 446 | 74 | IoT |
| 241 | 91 | NON_IoT |

In figure 6.5 we can see a piece of the neural network with one hidden layer with 26 neurons.

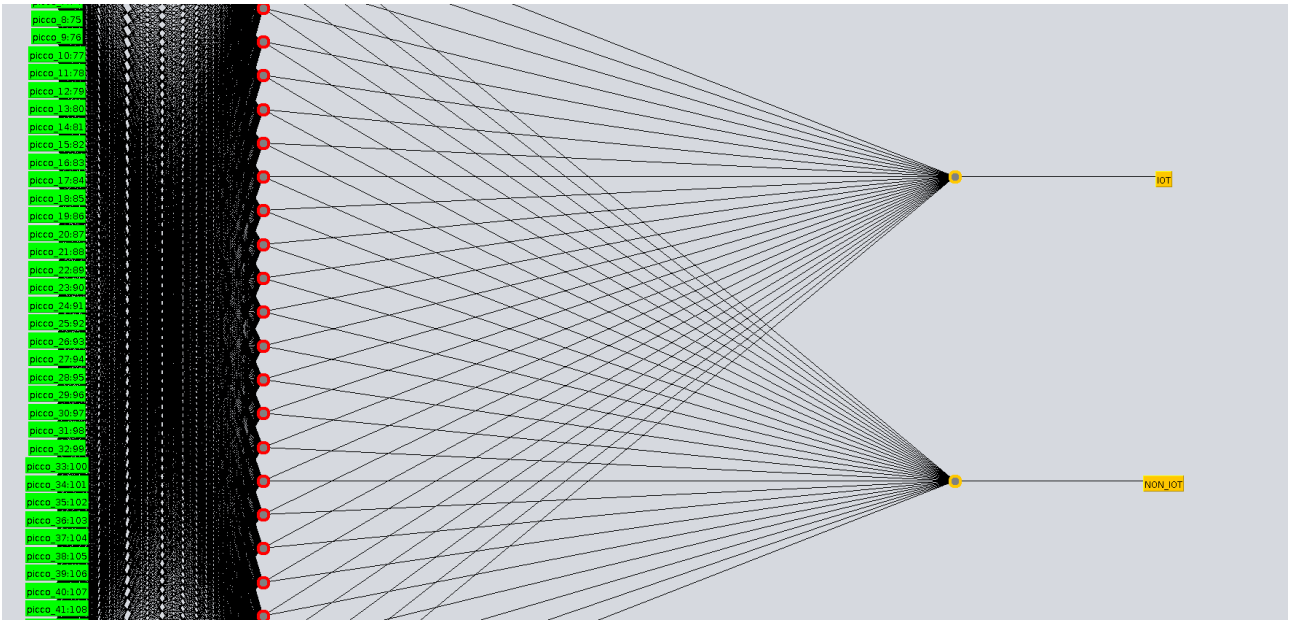



Figure 6.5: Neural network

### 6.2.7 Support Vector Machine

In a simple linear classifier, the boundaries between classes are straight lines, which are too simple for our application. A Support Vector Machine (SVM) uses linear models, however it transform the input space through a nonlinear mapping. Hence, the linear model in the new space is no longer linear when mapped back in the original space.

Table 6.14: Results of support vector machine

| Class | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 515 | 5 | 99.0% | 95.5% | 61.9% | 99.0% | 11.5% |
| NON_IoT | 15 | 317 | 4.5% | 1.0% | 75.0% | 4.5% | 11.5% |

Table 6.15: Results of support vector machine

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 530 - 62.20% | 322 - 37.80% | 852 - 100% |

The confusion matrix is:

Table 6.16: Results of support vector machine

| IoT | NON_IoT | Class |
|---|---|---|
| 515 | 5 | IoT |
| 317 | 15 | NON_IoT |

### 6.2.8 Naïve Bayes

The Naïve Bayes method is based on Bayes's rule and "naïvely" assumes independence of the attributes. The assumption that attributes are independent is simplistic, however the obtained classifier is simple to implement, and often works well when tested on actual datasets.

Table 6.17: Naïve Bayes Results

| Class | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 495 | 25 | 95.2% | 91.6% | 62.0% | 95.2% | 7.3% |
| NON_IoT | 28 | 304 | 8.4% | 4.8% | 52.8% | 8.4% | 7.3% |

Table 6.18: Naïve Bayes Results

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 523 - 61.38% | 329 - 38.62% | 852 - 100% |

The confusion matrix is:

Table 6.19: Naïve Bayes Results

| IoT | NON_IoT | Class |
|---|---|---|
| 495 | 25 | IoT |
| 304 | 28 | NON_IoT |

### 6.2.9 Euclidean Distance Between FFT points

In this section we introduce a new version of Weka [38] modified by Anilkumar Patro for the Data Mining class at WPI (Worcester Polytechnic Institute). This version of weka integrate the possibility to compute the FFT directly inside the software. Starting from this modified version of Weka, we tried another method to distinguish IoT flows from NON_IoT, which consists to calculate the euclidean distance between FFT points. The algorithm works in this way [42, 41]. Given a collection of time series CS, in our case the collection is the number of packet sampling, a collection of time series CT to be used as templates, a non-negative integer value r, and a non-negative real value epsilon. As described previously, given a time series S (in the time domain), it transforms the time series into a sequence FS (in the frequency domain) using the Fast Fourier Transformation and truncates this resulting sequence by keeping just the first r coefficients. Given two sequences, it computes the Euclidean distance between the sequences. For each template T in CT, it finds all the sequences in CS that are within an epsilon distance from T. This should be done following this procedure:

- All time series in CS and all the templates in CT are transformed and truncated as described above.
- For each transformed template FT:
  - For each transformed time series FS:
    - Compute the Euclidean distance between FT and FS.
    - If this distance is less than or equal to epsilon
      - Compute the Euclidean distance between T (the template before the transformation) and S (the time series before the transformation).
      - If this distance is less than or equal to epsilon then output the fact that T and S are "similar".

In our case we have done this procedure, first of all, we take in consideration all the IoT flows, and then we calculate the average distance between all points of it. We use this average as epsilon parameter. Then we calculate the average euclidean distance between a single non_IoT flow and all IoT flows, if the distance, is more than epsilon for at least one comparison, then the non_IoT flow is not correctly classified. We calculate the average euclidean distance between all the FFT points. In Cartesian coordinates, if p = (p1, p2,..., pn) and q = (q1, q2,..., qn) are two points in Euclidean n-space, then the distance (d) from p to q, or from q to p is given by the Pythagorean:

$$d(p,q) = \sqrt{\sum_{i=1}^{N}(q_i - p_i)^2} \tag{6.7}$$

### 6.2.10 Euclidean Distance Between FFT points Results

The total number of flows for this experiment is 631, 476 IoT, and 155 non_IoT. The number of IoT flows correctly classified are 416 out of 476 (87%), whilst the number of non_IoT correctly classified are 30 out of 155 (19%). We remind, an IoT flow is correctly classified only if the distances from the other IoT flows are less than epsilon, a non_IoT flow is correctly classified if all the distances from all IoT flows are greater or equal to epsilon. We tried to use differents epsilon parameters, but the results didn't improved. We denote that the IoT flows are almost correctly classified, instead for non_IoT flows performance are not good.

# 7 Packet Length Spectral Analysis

This part of our work is explained in our under review journal article [14]. In our approach, we construct a set of identifying features by computing the Fast Fourier Transform (FFT) spectrum of the series of packet lengths of a communication flow in the frequency domain. The basis of this technique was introduced by Liu et al., who consider the problem of classifying application traffic in the context of an encrypted network [53]. In our case, we apply the technique to distinguishing between IoT and non-IoT traffic, rather than to generic applications, and refine the method by considering alternatively the client to server, the server to client or the inter-arrival time as the source of data. This overcomes problems when the application does not transfer data, but only produces keep-alive messages.

The following subsections describe our approach in detail.

## 7.1 Related Work

Several methods have been proposed in the literature for packet classification, and we refer the reader to the existing literature for a systematic survey [35]. Here, we discuss the methods that are most closely related to our work.

The first aspect that must be considered when dealing with behavioral classification is an analysis of the communication pattern of the traffic that we want to classify. Shafiq et al. [44] have conducted a set of measurements to compare the machine-to-machine (M2M) traffic to traditional cellular smartphone traffic. The dataset comprises flows exchanged over the cellular data network in the USA. The first finding shows that M2M devices have a much larger uplink volume that downlink volume, in relative terms, compared to smartphones. This suggests a considerably different use of the network, and shows that M2M devices act more as content producers than consumers. The analysis also shows that M2M traffic follows business hours, and is significantly reduced in the weekends, as opposed to smartphone traffic which is virtually unchanged. Spectral analysis indicates strong periodicity in M2M traffic, corresponding to time intervals such as 1 hour, 30 minutes or 15 minutes, suggesting the timer-driven nature of M2M devices. Further analysis also reveals that devices are synchronized and coordinated. This may create congestion in the infrastructure. This frequency components are essentially absent from smartphone traffic. Sessions inter-arrival times are, instead, on average much longer for M2M devices than for smartphone traffic. An extensive list of discriminators for the purpose of flow classification is also presented by Moore et al. [32]. The dataset employed consists of general traffic captured for 24 hours at the authors' research facility. The report lists a total of 249 classes of discriminators, or features, that could be employed by a classifier.

In our previous work, we have explored the use of several of these discriminators, including round-trip time and fractions of uplink vs. downlink volume, through a decision tree classifier [19]. In this paper, we adopt a different classification strategy that makes use of an ensemble of decision trees (a random forest) to reduce the overfitting problem. At the same time, we considerably reduce the number of features, and focus on only the spectral analysis of the packet length of the data flow.

Among the early applications, a few techniques were developed to discriminate between Machine-to-Machine (M2M) traffic from remaining traffic (e.g., smartphones) specifically for cellular networks, using machine learning algorithms [13, 26]. These methods evaluate several features related to the traffic flow, including the number of packets, the data rate, the packet size, the inter packet-arrival time, as well as the IP address and TCP/UDP port numbers. Several derived features can also be computed from the flow data (averages, clusters, autocorrelations, etc.). Among the methods that are considered, there are unsupervised learning algorithms, which directly create classification, such as k-means, Expectation Maximization (EM) and Density-Based Spatial Clustering of Applications with Noise; and supervised methods, which require a learning phase with ground truth, such as J48,

Naive Bayesian (NB) and Support Vector Machine (SVM). In our previous work, we have employed neural networks as a supervised learning method [36]. Two interesting results can be learned from these studies. First, a limited number of features is sufficient to achieve a high level of classification accuracy. Second, Decision trees (such as J48) appear to be the machine learning algorithms that perform best in the studies.

Sivanathan et al. present a rich deployment consisting of a smart environment instrumented with 28 IoT devices, such as cameras, sensors, lights and smart plugs [48, 49, 47]. Traffic data was collected for six months, and then characterized in terms of several features, including employed protocols, data volume, port numbers and text patterns, activity and sleep cycles, and server queries. The authors present a two-stage hierarchical classifier, which uses a Naive Bayes Multinomial classifier to analyze domain names, port numbers and cypher suites, feeding a Random Forest classifier which integrates the remaining flow features. The combined approach is trained to recognize the different devices and shows a remarkable accuracy of 99.88%. A large part of our dataset is taken from this deployment, whose traffic data is made available in the public domain by the authors for download. Unlike this work, our aim is to classify traffic as traditional vs. IoT communication. More importantly, we focus on much fewer characteristics that capture the essence of an IoT device, rather than its specific implementation, and show that traffic patterns can be used as a distinguishing feature when analyzed in the frequency domain.

A similar approach is proposed by Meidan et al., who measure a deployment of nine different IoT devices connected in a network together with two PC's and two smartphones [17]. The authors employ features from the network, transport and application protocol layers, and classify the devices based first on a single communication session, and then on multiple sessions, outlining how the thresholds of the binary classifiers can be optimized to improve performance. While the reported classification accuracy is high (in excess of 99%), the authors do not discuss the way the features were selected, nor the kind of binary classifier and the way it is trained. This makes a comparison with other approaches problematic. As in the previous case, in our work we choose to ignore the protocol parameters to focus on the behavioral characteristics of the communication pattern, to abstract from the particular device employed.

An interesting approach is presented by Lopez-Martin et al., who apply a combination of a recurrent neural network (RNN) with a convolutional neural network (CNN) to perform traffic classification on a large dataset extracted from the Spanish academic backbone network [27]. For every flow, the authors consider the first 20 packets and collect a number of high-level header-based features, such as ports, window size, payload size and inter-arrival time. The main difference with respect to the previous approaches is that the features are organized in a time series, which is applied to the convolutional neural network as if it were an image, to identify local correlations. They then analyze the importance of each feature, by looking at the classification performance as they are added to or removed from the set. Overall, an accuracy up to 96% is reported for this work, on a dataset with over 100 different classification labels. In a different study, Yang et al. employ a Conditional Variational Autoencoder to address the problem of imbalance in intrusion detection systems, and use a 6-layer Deep Neural Network for classification [55]. As explained previously, our approach differs in the kind of features that we select from the dataset, as we ignore all the header data.

A number of solutions make use of ensemble learning to avoid overfitting and enhance the generalization power of the model. For instance, Shahid et al. extract features such as packet size and inter-arrival times of the first packets (10) of a flow from the network traffic of a smart home equipped with four devices [45]. The authors evaluate six different classification algorithms, and show that Random Forest performs best with an accuracy as high as 99.9%. One limitation of this study is the low number of devices and the use of data from the same deployment for both training and test, in the absence of non-IoT traffic. More recently, Amouri et al. have also employed a Random Forest classification algorithm in the context of intrusion detection systems [11]. Thangavelu et al. perform a similar study, focusing on scalability and the ability of the classifier to dynamically identify new devices [50]. This is accomplished by clustering the flows first with a standard k-means algorithm, then using a distributed semi-supervised clustering method by aggregating features. Clustering uses features such as DNS queries, number of packets, activity period in a session, TLS packet length,

flow duration, and number of packets of distinguished protocols such as DNS. The clusters are used as aggregate features to then periodically train a supervised learning algorithm for the final classification, which can therefore learn models for new devices. The approach is evaluated on an experimental setup consisting of 16 IoT devices for smart homes, monitored over a period of one week. The results show that a random forest classifier performs better than k-NN and Gaussian and Bernoulli naive Bayes with a 98% accuracy.

Finally, Pinheiro et al. present an approach based on a reduced set of features, relying essentially on the length of the packets (and their statistics, such as mean, mode and standard deviation) seen in a 1-second sampling window of the flow, to perform classification and reduce latency [40]. The selected features are independent of the specific header fields, so that classification can operate also in the presence of encryption. Classification is performed in three stages, distinguishing IoT and non-IoT traffic first, followed by the IoT device identification, and by the specific device event. Among five different classifiers, including k-NN, Random Forest, Decision Tree, SVM and Majority Voting, Random Forest is shown to perform best with accuracy reaching 96%. The approach is evaluated on a testbed composed of three IoT devices, complemented by traffic from the mentioned dataset of Sivanathan et al. [47]. Our approach is similar in spirit to this work, in that we rely on fewer attributes that characterize the behavior of the communication pattern, rather than the protocol data. Our novel contribution lies in the use of the frequency domain as an alternative space to perform the classification.

## 7.2 Dataset preparation and Baseline classification

Our analysis is based on a number of network captures codified as streams of packets encoded in standard `.pcap` files which are subsequently filtered by our software and analyzed by machine learning tools. Our main source of data is provided by a large dataset made available by the aforementioned work of Sivanathan et al. [48, 49, 47]. The data is obtained by capturing the traffic of a deployment of which we include 21 IoT devices and 7 non-IoT devices. We refer to this dataset as the *Australia dataset*.

Let we see how many flows per australian IoT device we have in our dataset (see the table 7.1).

In total we have 93478 IoT flows of australian dataset. Then we added other IoT flows provided by the USC/LANDER project [52, 51]. From this dataset, we extract traffic for 16 IoT devices, which were captured from a College Campus network. Although some of the devices are similar to those found in the first dataset, we do label them differently to account for potential different behaviors related to their specific use. In the following, we refer to this dataset as the *California dataset*.

Let we see how many flows per californian IoT device we have in our dataset (see the table 7.2).

In total we have 3421 IoT flows of californian dataset. As previous we introduce simulated IoT flows generated with MIMIC [5]. In this case we have a total of 6946 simulated IoT flows. We label this flows as "MIMIC".

Regarding the NON_IoT flows we use 19187 flows from Australian dataset for list of the devices see the table 4.2. To these, we add 2313 flows from an experiment from the University of New Brunswick [20](we to refer to this dataset as *canadian dataset*), and 10335 flows from a dataset collected by the Network Monitoring and Measurements research group at the University of Napoli [15, 16] (we refer to thi dataset as *napoli dataset*). Finally, we also include the "bigFlows" traffic dataset from Appneta Tcpreplay with 13847 flows [12] (we refer to this dataset as *bigflow dataset*). In total, there are 45682 non-IoT flows. Reccaping in table 7.4 we have:

### 7.2.1 Features computation

The procedure we follow to create the set of features is shown schematically in Figure 7.1. The data is processed directly from the `.pcap` file using a dedicated software built on top of `libpcap`. The analysis starts from a scan phase in which we identify and break up the flows, a check phase in which we select the packets, and a computation phase in which we compute the Fourier Transform of the data series. More specifically, in the first phase, we *scan* the flows to determine when to start the computation. The scan operation works through the `.pcap` file considering one packet at a time. The header is used to extract the timestamp and the length of the packet. The actual protocol headers in the packet data are used to extract flow information, in order to group packets into *flows* according to

Table 7.1: Australian IoT devices

| IoT Device | Number of flows |
|---|---|
| Smart Things | 32 |
| Amazon Echo | 3095 |
| Netatmo Welcome | 1809 |
| TP-Link Day Night Cloud camera | 851 |
| Samsung SmartCam | 5368 |
| Dropcam | 256 |
| Insteon Camera | 2998 |
| Withings Smart Baby Monitor | 4333 |
| Belkin Wemo switch | 6288 |
| TP-Link Smart plug | 167 |
| iHome | 149 |
| Belkin wemo motion sensor | 62440 |
| NEST Protect smoke alarm | 67 |
| Netatmo weather station | 1659 |
| Withings Smart scale | 28 |
| Withings Aura smart sleep sensor | 2532 |
| Light Bulbs LiFX Smart Bulb | 29 |
| Triby Speaker | 137 |
| PIX-STAR Photo-frame | 818 |
| HP Printer | 63 |
| Nest Dropcam | 359 |
| Total | 93478 |

Table 7.2: Californian IoT devices

| IoT Device | Number of flows |
|---|---|
| HP Printer California | 4 |
| TP-Link Smart plug California | 39 |
| Amazon Dash Bounty Button | 8 |
| Foscam IP CAM2 | 87 |
| Amazon Echo California | 639 |
| Google Smart Speaker | 185 |
| Amazon Fire SmartTVStick | 396 |
| Philips-Hue | 261 |
| AMCREST IP CAM | 337 |
| RENPHO Humidifier | 5 |
| Belkin Wemo switch California | 45 |
| TENVIS IP cam | 1 |
| D-link IP CAM | 331 |
| TP-Link SmartLightBulb California | 4 |
| Foscam IP CAM | 1035 |
| Wize IP CAM | 44 |
| Total | 3421 |

their source and destination, or client and server. A flow is constructed starting from the IP addresses, and the TCP flags, where the SYN flag denotes the beginning of the flow, and the FIN flag denotes its end. Each flow is forwarded to the next phase of computation whenever we reach a FIN packet (which identifies the end of the flows), or whenever we scan 256 packets from client to server, or if we scan 256 from server to client while at the same time we see at least 128 packets from client to server.

Table 7.3: MIMIC simulated IoT flows

| IoT Device | Number of flows |
|---|---|
| MIMIC | 6946 |

Table 7.4: Total flows IoT

| Dataset | Total number of flows |
|---|---|
| Australia IoT flows | 93478 |
| California IoT flows | 3421 |
| MIMIC IoT simulated flow | 6946 |
| Australia NON_IoT flows | 19187 |
| Canadian NON_IoT flows | 2313 |
| Napoli NON_IoT flows | 10335 |
| Bigflows NON_IoT flows | 13847 |
| Total IoT | 103845 |
| Total NON_IoT | 45682 |

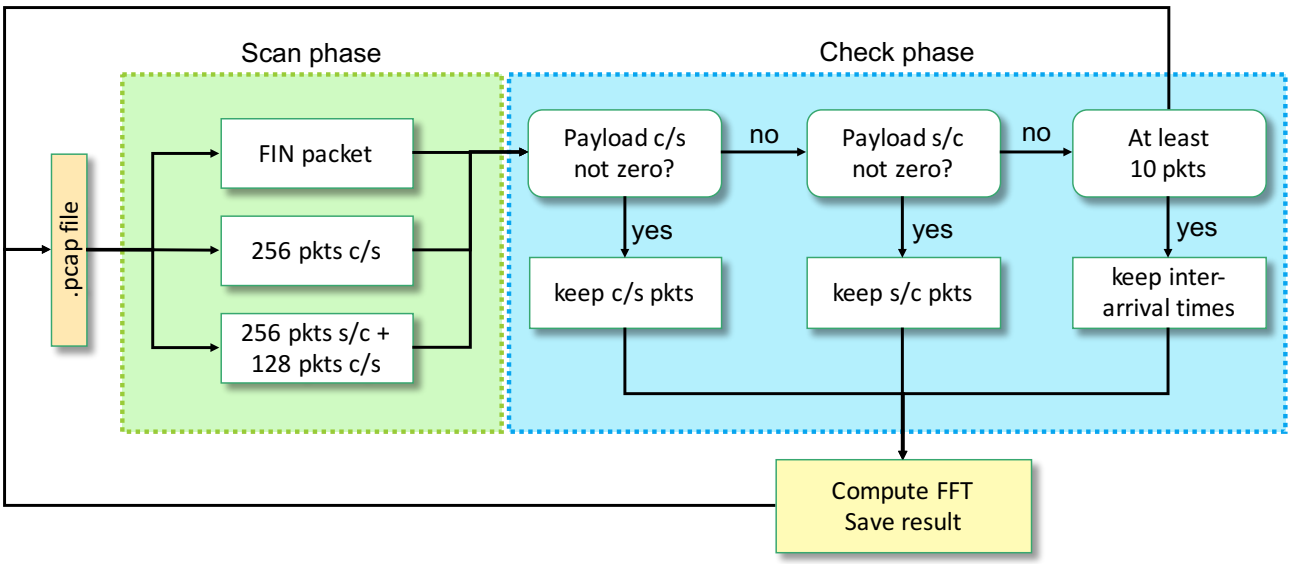



Figure 7.1: Data acquisition phases and features computation

The reason we do this is that we have found experimentally that the packets going from the client to the server provide a higher distinguishing power than the reverse direction. Note also that flows longer than 256 packets are broken up into different samples of length 256. This has the advantage of reducing the latency to obtain the classification results as well as its computational complexity. Conversely, flows that do not reach 256 packets at the FIN are padded with zeros to reach the desired length. This is necessary, as the FFT algorithm requires always the same number of input values.

During the second phase, we *check* the length of the packets for information content. For length, we denote the size of the payload, excluding the base protocol headers (IP, TCP and UDP). If the sequence of packets from client to server consists of only empty payloads, we check the packets in the reverse direction. If these are also empty, then we use the packet inter-arrival times, computed from their timestamps, of the client to server packets as the data, since the payload provides no useful information, while the periodicity is used as a discriminator.

The FFT is computed on the selected data by a dedicated software library [4] and produces a symmetric spectrum of 256 elements, of which only the right 128 values are retained. The computation

follows the traditional formulation

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N},$$

where $x_n$ are the selected packet payload lengths in the flow, $N$ is the number of samples (256 in our case), and $X_k$ are the spectrum output coefficients. In particular, we use the magnitude $||X_k||^2$ of the spectrum, which is proportional to the power content at each specific frequency. The computation of the spectrum resembles the application of the first layers of a convolutional neural network, except that the filter are fixed and not learned from the data. This, on the other hand, simplifies both training and inference. Having characterized each flow with its FFT, we label the data by matching the Ethernet MAC address of the packets to those of the devices present in the network. The FFT coefficients and the labels are then assembled into an `.arff` file that is given as input to Weka [9], which in turn partitions the data into training, validation and test sets and performs the model optimization. At the end of the process, the tool provides the performance metrics from cross validation and testing, as discussed in the next section.

## 7.3 Results

In this section, we summarize the results obtained by training a Random Forest classifier with the data acquired using the procedure outlined in the previous section. We evaluate the classification accuracy using 10-fold cross validation. With this method, the dataset is divided into 10 random subsets, which are alternatively used for training and for determining the classification accuracy. The results from ten different rounds are then averaged to give the final metrics. Because our dataset is somewhat skewed towards the IoT class, an evaluation based on accuracy alone, i.e., the ratio between the correctly classified flows and the total number of flows, is unable to provide a proper picture of the performance of the classifier. For this reason, in addition to True Positives, False Positives and False Negatives, we employ the following metrics [21]:

- Precision: for a given class, it is the ratio of its True Positives and the sum of True Positives and False Positive (i.e., a sample of another class that is labeled as one of this class).
- Recall: for a given class, it is the ratio of the True Positives and the sum of True Positives and False Negatives (i.e., a sample of this class is labeled as not of this class).
- Matthews correlation coefficient (MCC): the MCC returns a value between -1 and +1, where +1 represents a perfect prediction, 0 is no better than a random prediction and -1 indicates total disagreement between prediction and observation.

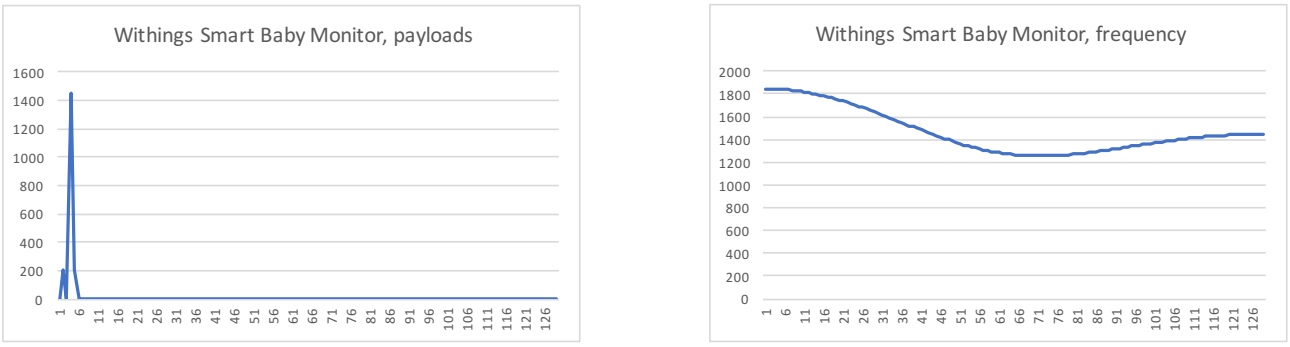
The Matthews correlation coefficient measure is particularly significant for binary classification (for instance, when we distinguish between IoT and non-IoT classes) especially when the classes are of different size as in our case. In addition to these, one alternative measure is the F-Measure, a widely used metric in classification, which weighs both Precision and Recall in a single metric by taking the harmonic mean: 2 x Recall x Precision / (Recall + Precision). When running the 10-fold validation procedure, the Weka tool automatically returns these measures averaged over the different iterations of the process.

### 7.3.1 Data visualization

It is useful to visualize the data features that correspond to the packet flows of different devices, to appreciate the distinguishing power of the transformed signal. In this section we discuss a few examples, where we show and compare the time series as well as the frequency spectra of selected flows.

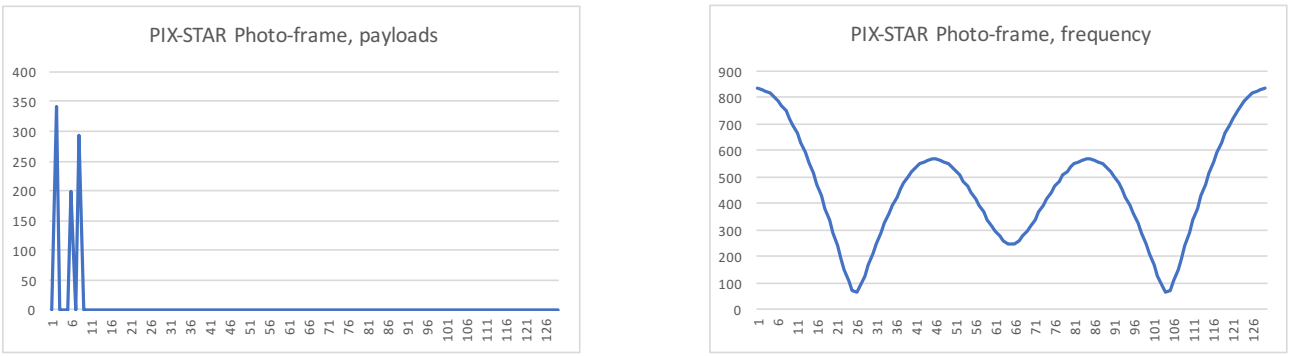
Figure 7.2 to Figure 7.8 show both the series of the payload (on the left) and its FFT, limited to one side of the symmetric magnitude of the spectrum (on the right). We see that for short flows, the "energy" of the packets is spread across the entire spectrum with a shape that depends on the number of peaks that are found in the time series. These cases are shown in Figure 7.2 and Figure 7.3. The energy in the spectrum, i.e., the magnitude of each frequency component, depends on the size of

the payloads in the flow, and this constitutes another difference than can be used by the classifier to distinguish the flows. This is true in all the examples shown.




**a** **b**

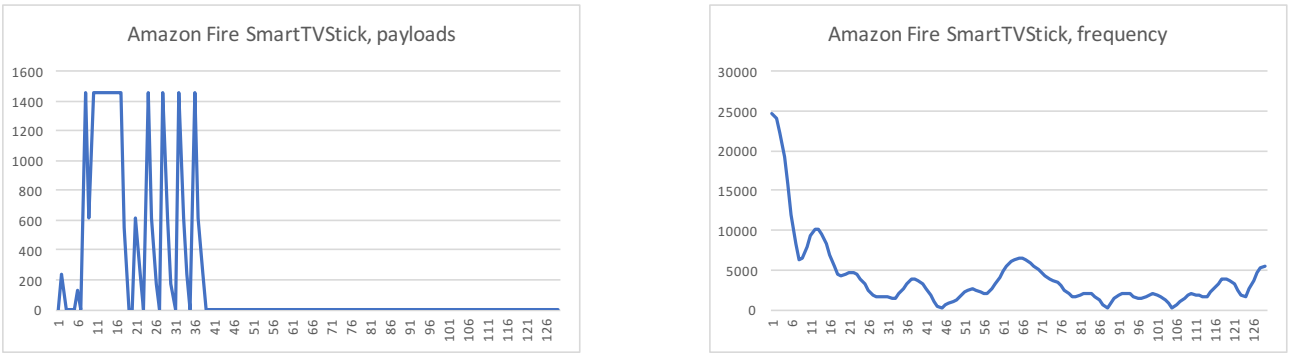
Figure 7.2: Withings Baby monitor: (**a**) Payload time series. (**b**) Frequency series.




**a** **b**

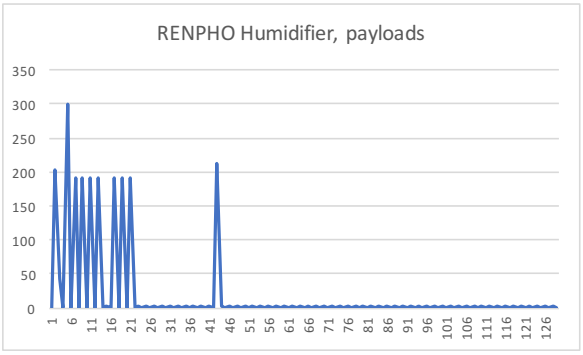
Figure 7.3: PIX-STAR Photo Frame: (**a**) Payload time series. (**b**) Frequency series.

Figure 7.4 and Figure 7.5 show the spectrum of somewhat longer payload series, with limited regularities. These kind of spectra are also spread out, but are more concentrated around certain specific frequencies that distinguish the nature of the traffic. On the other hand, traffic that is characterized by strong regularities gives rise to precise peaks in the corresponding spectrum. This case is shown for instance in Figure 7.6, where the spectrum shows peaks at very specific frequencies, on top of a noisy background that depends on the small variabilities in the data. The shape of the spectrum is clearly very different compared to the previous cases.

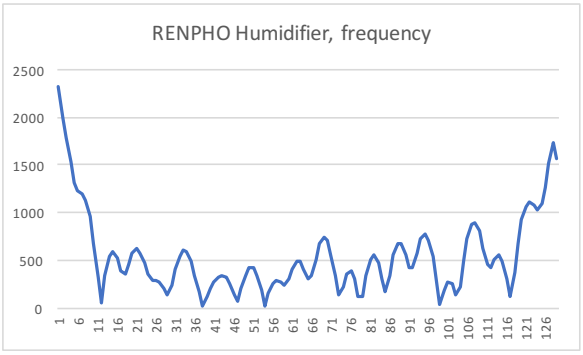



**a** **b**

Figure 7.4: Amazon Fire SmartTVStick: (**a**) Payload time series. (**b**) Frequency series.

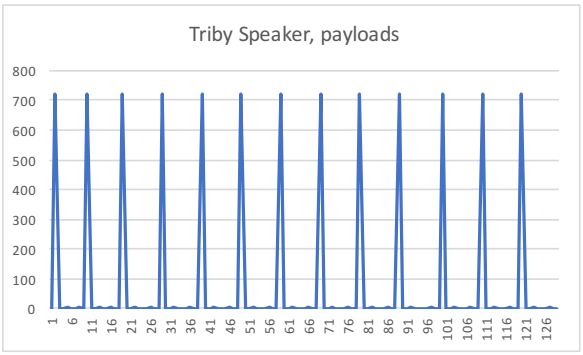



**a**

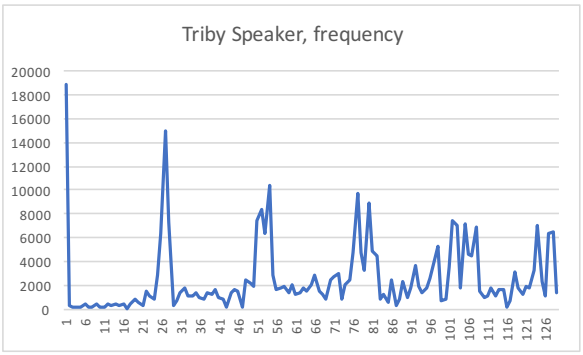



**b**

Figure 7.5: RENPHO Humidifier: (**a**) Payload time series. (**b**) Frequency series.



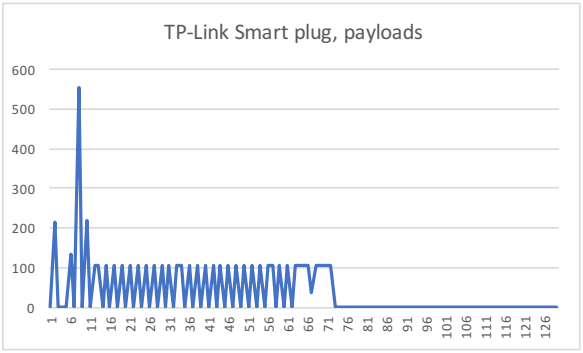

**a**



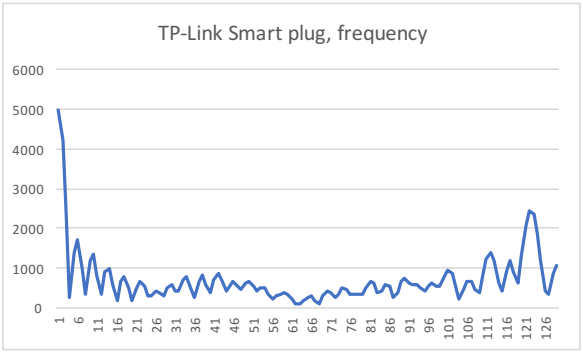

**b**

Figure 7.6: Triby Speaker: (**a**) Payload time series. (**b**) Frequency series.

Finally, longer payload series with recurring and closer regularities result in denser peaks in the spectrum, as shown in Figure 7.7 and to a lesser extent in Figure 7.8. This last example is more difficult to characterize in terms of a definitive shape, but at the same time can be easily set apart from the more distinctive patterns exhibited by the other devices.

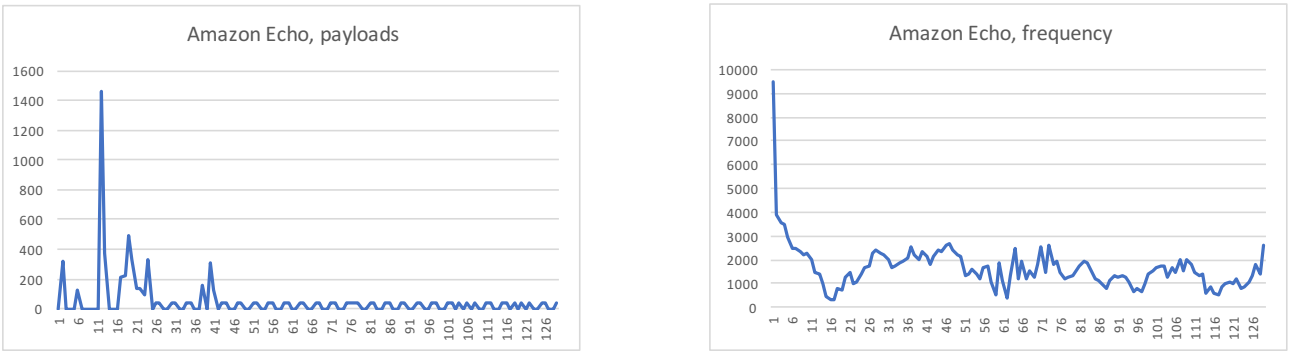



**a**




**b**

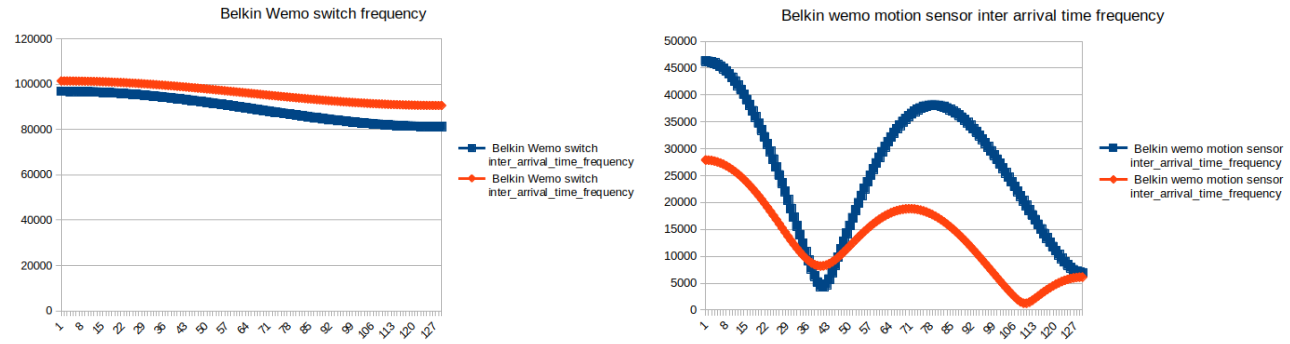
Figure 7.7: TP-Link Smart Plug: (**a**) Payload time series. (**b**) Frequency series.




**a** **b**

Figure 7.8: Amazon Echo: (**a**) Payload time series. (**b**) Frequency series.

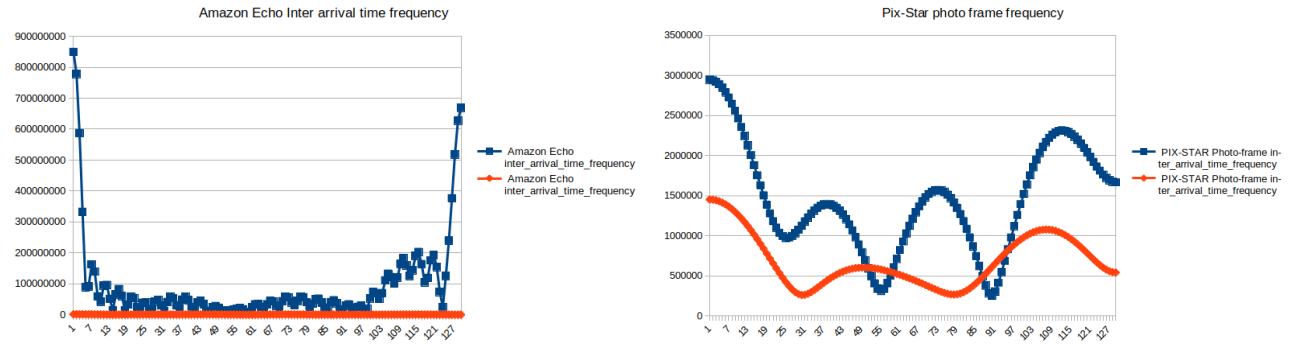
Then we tried to visualize graphically the inter-arrival series. The inter-arrival time series are very variable, this is the reason for which the classification performs worse with this feature. In figure 7.9 we see the differences of inter-arrival time between flows generated by the same devices. For this two devices we can see that the difference are not so substantial.




**a** **b**

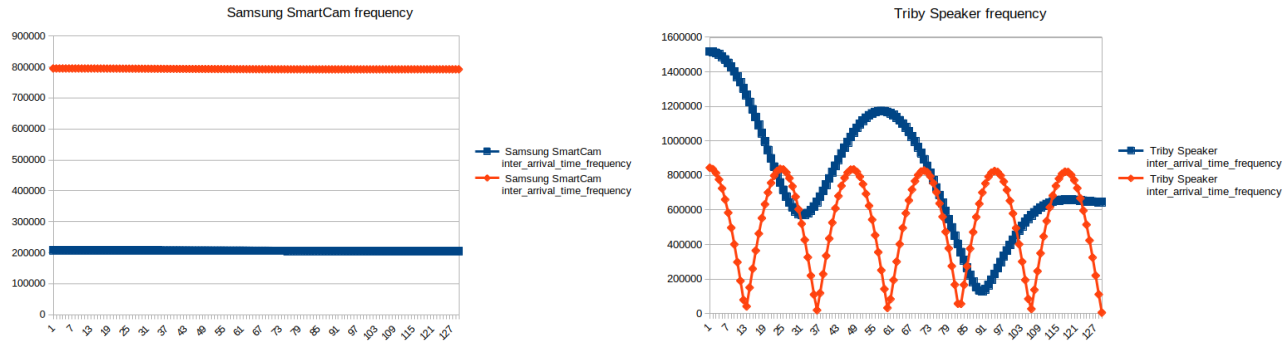
Figure 7.9: (**a**) Belkin wemo motion sensor. (**b**) Belkin wemo switch.

Furthermore, in figures 7.10 7.11 7.12 we see other inter-arrival time frequencies.



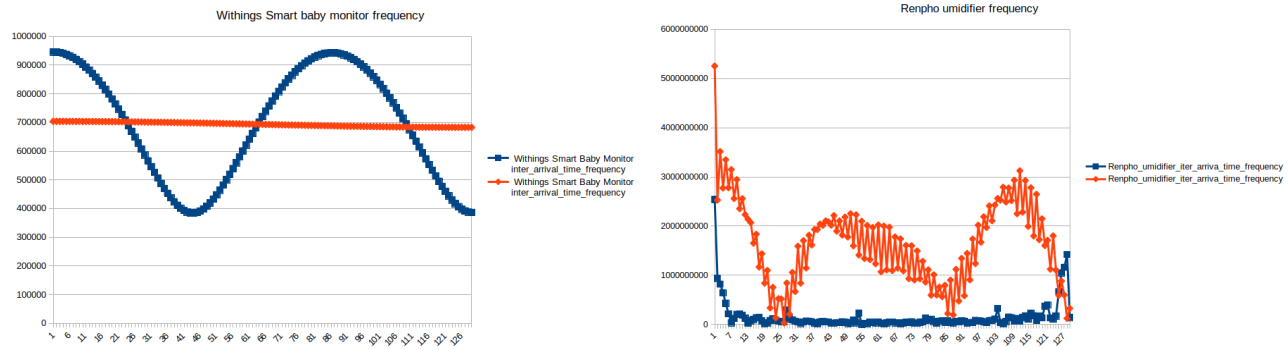

**a** **b**

Figure 7.10: (**a**) Amazon Echo. (**b**) Pix-star photo frame.




**a** **b**

Figure 7.11: (**a**) Samsung Smartcam. (**b**) Triby speaker




**a** **b**

Figure 7.12: (**a**) Withings Smart baby monitor. (**b**) Renpho umidifier

### 7.3.2 Classification results

In this section we present the results of the classification algorithm evaluated using the 10-fold cross validation technique. For classification, we have used a Random Forest classifier. This classifier is an example of an *ensemble* algorithm: instead of a giant decision tree, which may easily overfit the data, the Random Forest is composed of a number of simpler decision trees, each providing its own classification. The overall results corresponds to the majority voting of the individual trees. Because the trees are simpler, they also result in a simpler hypothesis function, reducing the overfitting problem. In our case, we have instructed the Weka tool to construct a Random Forest with 100 trees.

We have divided the evaluation into two main parts, each composed of three different mixes of the dataset. In the first part, we evaluate in particular the ability of the classifier to distinguish among the different IoT devices. Each flow in the dataset is therefore labeled with the device name, and the classifier is trained to distinguish the individual labels. In a second set of experimental evaluation we introduce also the traditional non-IoT traffic. In this case, we evaluate both a classifier that is able to distinguish among the different devices, as well as a classifier that simply distinguishes between the IoT and the non-IoT class. The following two sections present the details of the evaluation.

### 7.3.3 Classification of individual devices

We train three different classifiers to evaluate the classification performance on different datasets. In the first two cases we look at the classifier using the Australia and the California datasets individually. The third classifier is instead trained on the combined dataset, including the simulated flows from the MIMIC Simulator. The results for these experiments in terms of Precision, Recall and MCC are shown in Tables 7.5, 7.8 and 7.11, while Tables 7.7, 7.10 and 7.12 show the corresponding accuracy. We show also the different confusion matrix in tables 7.6  7.9 The results are especially positive for the Australia dataset, which comprises the majority of the flows. In particular, the False Positive rate is very low, and the MCC is close to 1, indicating a high degree of reliability (see Table 7.5). The overall accuracy for this dataset come in excess of 99% (Table 7.7).

The California dataset, which is much smaller in size, gives a mixed outcome when considered alone. The break-up of the classification results in Table 7.8 shows that a few of the devices are difficult to recognize, especially when the corresponding number of flows in the training set is particularly low. In this case, the 10-fold cross validation may at times fail to include the flows in the actual training set. This indicates that the classifier needs a sufficient number of examples to correctly identify devices which have a low utilization of the network. The overall accuracy, in this case is much lower at 86.52% and the MCC only achieves a value of 84.2%.

Table 7.5: Classification performance for the Australia dataset, IoT only

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| Dropcam | 235 | 21 | 91.8% | 0.0% | 88.0% | 91.8% | 89.9% |
| Smart Things | 23 | 9 | 71.9% | 0.0% | 100% | 71.9% | 84.8% |
| Withings Smart baby monitor | 4328 | 5 | 99.9% | 0.0% | 99.9% | 99.9% | 99.9% |
| Belkin Wemo Motion Sensor | 62413 | 27 | 100.0% | 1.6% | 99.2% | 100.0% | 98.7% |
| Samsung SmartCam | 5357 | 11 | 99.8% | 0.0% | 99.4% | 99.8% | 99.6% |
| Belkin_Wemo_Switch | 6253 | 35 | 99.4% | 0.0% | 99.7% | 99.4% | 99.5% |
| PIX-STAR Photo frame | 812 | 6 | 99.3% | 0.0% | 99.3% | 99.3% | 99.3% |
| Amazon_Echo | 3028 | 67 | 97.8% | 0.1% | 98.0% | 97.8% | 97.8% |
| TP-Link Smart Plug | 166 | 1 | 99.4% | 0.0% | 96.5% | 99.4% | 97.9% |
| Netatmo weather station | 1657 | 2 | 99.9% | 0.0% | 99.9% | 99.9% | 99.9% |
| TP-Link DayNight CloudCam | 819 | 32 | 96.2% | 0.0% | 99.4% | 96.2% | 97.8% |
| Netatmo Welcome | 1796 | 13 | 99.3% | 0.0% | 98.5% | 99.3% | 98.9% |
| Withings Smart Scale | 26 | 2 | 92.9% | 0.0% | 100% | 92.9% | 96.4% |
| Triby Speaker | 122 | 15 | 89.1% | 0.0% | 93.8% | 89.1% | 91.4% |
| NEST Protect smoke alarm | 60 | 7 | 89.6% | 0.0% | 100% | 89.6% | 94.6% |
| HP printer | 61 | 2 | 96.8% | 0.0% | 98.4% | 96.8% | 97.6% |
| Insteon Camera | 2998 | 0 | 100% | 0.0% | 99.8% | 100% | 99.9% |
| Withings AuraSmartSleepSensor | 2062 | 470 | 81.4% | 0.0% | 99.1% | 81.4% | 89.6% |
| iHome | 139 | 10 | 93.3% | 0.0% | 98.6% | 93.3% | 95.9% |
| Light Bulbs LiFX SmartBulb | 23 | 6 | 79.3% | 0.0% | 95.8% | 79.3% | 87.2% |
| Nest Dropcam | 336 | 23 | 93.6% | 0.0% | 93.1% | 93.6% | 93.3% |
| **Weighted Avg.** | **92714** | **764** | **99.2%** | **1.1%** | **99.2%** | **99.2%** | **98.6%** |

Table 7.6: Confusion matrix for the Australia dataset, IoT only

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | Classified as |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 235 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 1 | 11 | a = Dropcam |
| 8 | 23 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | b = Smart Things |
| 0 | 0 | 4328 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | c = Withings Smart baby monitor |
| 0 | 0 | 0 | 62413 | 2 | 18 | 0 | 5 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d = Belkin Wemo Motion Sensor |
| 1 | 0 | 0 | 1 | 5357 | 0 | 0 | 4 | 3 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | e = Samsung SmartCam |
| 0 | 0 | 0 | 25 | 0 | 6253 | 0 | 5 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | f = Belkin_Wemo_Switch |
| 0 | 0 | 0 | 0 | 0 | 0 | 812 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | g = PIX-STAR Photo frame |
| 1 | 0 | 0 | 10 | 25 | 0 | 1 | 3028 | 0 | 0 | 1 | 7 | 0 | 5 | 0 | 0 | 6 | 2 | 0 | 0 | 9 | h = Amazon_Echo |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 166 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | i = TP-Link Smart Plug |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1657 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | j = Netatmo weather station |
| 14 | 0 | 0 | 1 | 3 | 0 | 1 | 6 | 3 | 1 | 819 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | k = TP-Link DayNight CloudCam |
| 0 | 0 | 0 | 6 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 1796 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 2 | l = Netatmo Welcome |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | m = Withings Smart Scale |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 8 | 0 | 1 | 0 | 1 | 0 | 122 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | n = Triby Speaker |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 4 | 0 | 0 | 0 | 2 | 0 | 0 | 60 | 0 | 0 | 0 | 0 | 0 | 0 | o = NEST Protect smoke alarm |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 61 | 0 | 0 | 0 | 0 | 0 | p = HP printer |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2998 | 0 | 0 | 0 | 0 | q = Insteon Camera |
| 0 | 0 | 4 | 461 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2062 | 1 | 0 | 1 | r = Withings AuraSmartSleepSensor |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 139 | 0 | 0 | s = iHome |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 23 | 0 | t = Light Bulbs LiFX SmartBulb |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 336 | u = Nest Dropcam |

Table 7.7: Australia dataset IoT only: overall accuracy

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 92714 - 99.18% | 764 - 0.82% | 93478 - 100% |

Table 7.8: Classification performance for the California dataset, IoT only

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| HP printer | 2 | 2 | 50.0% | 0.1% | 50.0% | 50.0% | 49.9% |
| Amazon_Dash_Bounty_Button | 8 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo | 622 | 17 | 97.3% | 7.50% | 74.8% | 97.3% | 81.6% |
| Amazon_FireTVStick | 200 | 196 | 50.5% | 2.2% | 74.6% | 50.5% | 57.5% |
| AMCREST_IPCAM | 284 | 53 | 84.3% | 1.3% | 87.4% | 84.3% | 84.3% |
| Belkin_Wemo_Switch | 30 | 15 | 66.7% | 0.2% | 78.9% | 66.7% | 72.2% |
| D-link_IPCAM | 304 | 27 | 91.8% | 0.8% | 92.1% | 91.8% | 91.1% |
| FOSCAM_IPCAM | 978 | 57 | 94.5% | 1.5% | 96.5% | 94.5% | 93.6% |
| FOSCAM_IPCAM_vers2 | 52 | 35 | 59.8% | 0.3% | 83.9% | 59.8% | 70.2% |
| Google_Smartspeaker | 169 | 16 | 91.4% | 1.1% | 82.8% | 91.4% | 86.2% |
| Philips_Hue | 249 | 12 | 95.4% | 0.4% | 95.4% | 95.4% | 95.0% |
| RENPHO_humidifier | 0 | 5 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug | 32 | 7 | 82.1% | 0.2% | 84.2% | 82.1% | 82.9% |
| TP-Link SmartLightBulb | 2 | 2 | 50.0% | 0.1% | 50.0% | 50.0% | 49.9% |
| Wyze_IPCAM | 28 | 16 | 63.6% | 0.1% | 84.8% | 63.6% | 73.2% |
| **Weighted Avg.** | **2960** | **461** | **86.5%** | **2.4%** | **86.6%** | **86.5%** | **84.2%** |

Table 7.9: Confusion matrix for the California dataset, IoT only

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | Classified as |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | a = HP_Printer |
| 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | b = Amazon_Dash_Bounty_Button |
| 0 | 0 | 622 | 15 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | c = Amazon_Echo |
| 1 | 0 | 150 | 200 | 6 | 3 | 7 | 5 | 2 | 13 | 1 | 0 | 0 | 4 | 0 | 4 | d = Amazon_FireTVStick |
| 0 | 0 | 14 | 6 | 284 | 0 | 7 | 17 | 4 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | e = AMCREST_IPCAM |
| 0 | 0 | 2 | 5 | 0 | 30 | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | f = Belkin_Wemo_Switch |
| 0 | 0 | 5 | 5 | 6 | 2 | 304 | 3 | 2 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | g = D-link_IPCAM |
| 0 | 0 | 23 | 3 | 18 | 0 | 5 | 978 | 2 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | h = FOSCAM_IPCAM |
| 1 | 0 | 8 | 13 | 1 | 0 | 1 | 1 | 52 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | i = FOSCAM_IPCAM_vers2 |
| 0 | 0 | 2 | 3 | 3 | 2 | 1 | 2 | 0 | 169 | 0 | 0 | 0 | 1 | 1 | 1 | j = Google_Smartspeaker |
| 0 | 0 | 0 | 5 | 4 | 0 | 0 | 1 | 0 | 2 | 249 | 0 | 0 | 0 | 0 | 0 | k = Philips_Hue |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | l = RENPHO_humidifier |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | m = TENVIS_IPCAM |
| 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 0 | n = TP-Link Smart Plug |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | o = TP-Link SmartLightBulb |
| 0 | 0 | 6 | 4 | 2 | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 28 | p = Wyze_IPCAM |

Table 7.10: California dataset IoT only: overall accuracy

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 2960 - 86.52% | 461 - 13.48% | 3421 - 100% |

When combined (Table 7.11 and 7.12) the overall classification performance is acceptable, although the data is obviously dominated by the much larger Australia dataset. In this case, we observe a slight

increase in False Positive rate, although the weighted average fails to show this because the weight of each device, especially those for which recognition is harder, is much lower with the large size of the dataset. The tables also show that the algorithm performs extremely well on the MIMIC simulated flows. These flows are clearly more homogeneous, indicating that traffic captured from real devices is essential in the context of classification.

Table 7.11: Classification performance of the combined dataset, IoT only

| **Device** | **Correct** | **Wrong** | **TP rate** | **FP rate** | **Precision** | **Recall** | **MCC** |
|---|---|---|---|---|---|---|---|
| Dropcam | 235 | 21 | 91.8% | 0.0% | 88.0% | 91.8% | 89.9% |
| Smart Things | 27 | 5 | 84.4% | 0.0% | 100% | 84.4% | 91.9% |
| Withings Smart baby monitor | 4327 | 6 | 99.9% | 0.0% | 99.6% | 99.9% | 99.7% |
| Belkin Wemo Motion Sensor | 62413 | 27 | 100.0% | 1.3% | 99.2% | 100.0% | 98.9% |
| Samsung SmartCam | 5355 | 13 | 99.8% | 0.0% | 99.3% | 99.8% | 99.5% |
| Belkin_Wemo_Switch | 6252 | 36 | 99.4% | 0.0% | 99.7% | 99.4% | 99.5% |
| PIX-STAR Photo frame | 810 | 8 | 99.0% | 0.0% | 98.1% | 99.0% | 98.5% |
| Amazon_Echo | 3021 | 74 | 97.6% | 0.2% | 95.1% | 97.6% | 96.2% |
| TP-Link Smart Plug | 166 | 1 | 99.4% | 0.0% | 96.5% | 99.4% | 97.9% |
| Netatmo weather station | 1656 | 3 | 99.8% | 0.0% | 99.4% | 99.8% | 99.6% |
| TP-Link DayNight CloudCam | 818 | 33 | 96.1% | 0.0% | 99.2% | 96.1% | 97.6% |
| Netatmo Welcome | 1796 | 13 | 99.3% | 0.0% | 97.4% | 99.3% | 98.3% |
| Withings Smart Scale | 23 | 5 | 82.1% | 0.0% | 100% | 82.1% | 90.6% |
| Triby Speaker | 121 | 16 | 88.3% | 0.0% | 87.7% | 88.3% | 88.0% |
| NEST Protect smoke alarm | 61 | 6 | 91.0% | 0.0% | 98.4% | 91.0% | 94.6% |
| HP printer | 57 | 6 | 90.5% | 0.0% | 95.0% | 90.5% | 92.7% |
| Insteon Camera | 2998 | 0 | 100% | 0.0% | 99.8% | 100% | 99.9% |
| Withings AuraSmartSleepSensor | 2045 | 487 | 80.8% | 0.0% | 97.6% | 80.8% | 88.5% |
| iHome | 136 | 13 | 91.3% | 0.0% | 97.8% | 91.3% | 94.5% |
| Light Bulbs LiFX SmartBulb | 18 | 11 | 62.1% | 0.0% | 94.7% | 62.1% | 76.7% |
| Nest Dropcam | 327 | 32 | 91.1% | 0.0% | 90.6% | 91.1% | 90.8% |
| MIMIC | 6945 | 1 | 100% | 0.0% | 99.8% | 100% | 99.9% |
| HP printer Californian | 2 | 2 | 50.0% | 0.0% | 40.0% | 50.0% | 44.7% |
| Amazon_Dash_Bounty_Button | 8 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo Californian | 601 | 38 | 94.1% | 0.2% | 73.6% | 94.1% | 83.1% |
| Amazon_FireTVStick | 138 | 258 | 34.8% | 0.0% | 73.8% | 34.8% | 50.6% |
| AMCREST_IPCAM | 259 | 78 | 76.9% | 0.0% | 84.6% | 76.9% | 80.6% |
| Belkin_Wemo_Switch_Californian | 27 | 18 | 60.0% | 0.0% | 84.4% | 60.0% | 71.1% |
| D-link_IPCAM | 301 | 30 | 90.9% | 0.0% | 90.1% | 90.9% | 90.5% |
| FOSCAM_IPCAM | 955 | 80 | 92.3% | 0.0% | 96.8% | 92.3% | 94.4% |
| FOSCAM_IPCAM_vers2 | 50 | 37 | 57.5% | 0.0% | 86.2% | 57.5% | 70.4% |
| Google_Smartspeaker | 146 | 39 | 78.9% | 0.0% | 80.2% | 78.9% | 79.5% |
| Philips_Hue | 233 | 28 | 89.3% | 0.0% | 95.9% | 89.3% | 92.5% |
| RENPHO_humidifier | 0 | 5 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug Californian | 34 | 5 | 87.2% | 0.0% | 82.9% | 87.2% | 85.0% |
| TP-Link SmartLightBulb | 0 | 4 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| Wyze_IPCAM | 19 | 25 | 43.2% | 0.0% | 90.5% | 43.2% | 62.5% |
| **Weighted Avg.** | **102380** | **1465** | **98.6%** | **0.8%** | **98.6%** | **98.6%** | **98.1%** |

Table 7.12: Combined dataset IoT only: overall accuracy

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 102380 - 98.59% | 1465 - 1.41% | 103845 - 100% |

### Traditional vs. IoT traffic classification

In this set of experiments we have trained the classifier to recognize the IoT devices together with the non-IoT traffic lumped into a single class. The addition of the non-IoT traffic slightly degrades the classification accuracy. Tables 7.13 7.14 shows the results, which indicate that the non-IoT flows can be well separated from the individual devices. The overall accuracy is $137133/(137133+2027) = 98.54\%$.

Table 7.13: Australia IoT with non-IoT traffic

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| Dropcam | 193 | 63 | 75.4% | 0.0% | 96.0% | 75.4% | 85.1% |
| Smart Things | 24 | 8 | 75.0% | 0.0% | 100% | 75.0% | 86.6% |
| Withings Smart baby monitor | 4303 | 30 | 99.3% | 0.2% | 95.1% | 99.3% | 97.1% |
| Belkin Wemo Motion Sensor | 62395 | 45 | 99.9% | 0.8% | 99.1% | 99.9% | 99.1% |
| Samsung SmartCam | 5353 | 15 | 99.7% | 0.0% | 99.2% | 99.7% | 99.4% |
| Belkin_Wemo_Switch | 6247 | 41 | 99.3% | 0.0% | 99.8% | 99.3% | 99.5% |
| PIX-STAR Photo frame | 794 | 24 | 97.1% | 0.0% | 99.3% | 97.1% | 98.1% |
| Amazon_Echo | 2907 | 188 | 93.9% | 0.1% | 97.1% | 93.9% | 95.4% |
| TP-Link Smart Plug | 153 | 14 | 91.6% | 0.0% | 100.0% | 91.6% | 95.7% |
| Netatmo weather station | 1653 | 6 | 99.6% | 0.0% | 100.0% | 99.6% | 99.8% |
| TP-Link DayNight CloudCam | 802 | 49 | 94.2% | 0.0% | 99.4% | 94.2% | 96.8% |
| Netatmo Welcome | 1770 | 39 | 97.8% | 0.0% | 99.7% | 97.8% | 98.8% |
| Withings Smart Scale | 25 | 3 | 89.3% | 0.0% | 100% | 89.3% | 94.5% |
| Triby Speaker | 88 | 49 | 64.2% | 0.0% | 91.7% | 64.2% | 76.7% |
| NEST Protect smoke alarm | 50 | 17 | 74.6% | 0.0% | 100% | 74.6% | 86.4% |
| HP printer | 52 | 11 | 82.5% | 0.0% | 100.0% | 82.5% | 90.8% |
| Insteon Camera | 2998 | 0 | 100% | 0.0% | 99.7% | 100% | 99.9% |
| Withings AuraSmartSleepSensor | 1862 | 670 | 73.5% | 0.1% | 91.2% | 73.5% | 81.6% |
| iHome | 134 | 15 | 89.9% | 0.0% | 99.3% | 89.9% | 94.5% |
| Light Bulbs LiFX SmartBulb | 0 | 29 | 0.0% | 0.0% | - | 0.0% | - |
| Nest Dropcam | 243 | 116 | 67.7% | 0.0% | 97.2% | 67.7% | 81.1% |
| NON_IoT | 45087 | 595 | 98.7% | 0.9% | 98.1% | 98.7% | 97.6% |
| **Weighted Avg.** | **137133** | **2027** | **98.5%** | **0.6%** | **98.5%** | **98.5%** | **98.0%** |

Table 7.14: Confusion matrix Australia IoT with non-IoT traffic

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | Classified as |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 193 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 58 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | a = Dropcam |
| 6 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | b = Smart Things |
| 0 | 0 | 4303 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 28 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | c = Withings Smart Baby Monitor |
| 0 | 0 | 0 | 62395 | 2 | 14 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d = Belkin wemo motion sensor |
| 0 | 0 | 0 | 0 | 5353 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | e = Samsung SmartCam |
| 0 | 0 | 0 | 18 | 0 | 6247 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | f = Belkin Wemo switch |
| 0 | 0 | 0 | 0 | 0 | 0 | 794 | 0 | 0 | 0 | 0 | 0 | 0 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | g = PIX-STAR Photo-frame |
| 0 | 0 | 0 | 10 | 25 | 0 | 0 | 2907 | 0 | 0 | 0 | 1 | 0 | 145 | 1 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | h = Amazon Echo |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 153 | 0 | 0 | 0 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | i = TP-Link Smart plug |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1653 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | j = Netatmo weather station |
| 0 | 0 | 0 | 0 | 3 | 0 | 0 | 1 | 0 | 0 | 802 | 0 | 0 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | k = TP-Link Day Night Cloud camera |
| 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1770 | 0 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | l = Netatmo Welcome |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | m = Withings Smart scale |
| 0 | 0 | 219 | 83 | 12 | 1 | 6 | 77 | 0 | 0 | 5 | 4 | 0 | 45087 | 7 | 0 | 0 | 2 | 177 | 0 | 0 | 2 | 0 | n = NON_IOT |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 44 | 88 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | o = Triby Speaker |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | p = NEST Protect smoke alarm |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 52 | 0 | 0 | 0 | 0 | 0 | q = HP Printer |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2998 | 0 | 0 | 0 | 0 | r = Insteon Camera |
| 0 | 0 | 3 | 461 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 204 | 0 | 0 | 0 | 0 | 1862 | 1 | 0 | 0 | s = Withings Aura smart sleep sensor |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 134 | 0 | 0 | t = iHome |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | u = Light Bulbs LiFX Smart Bulb |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 114 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 243 | v = Nest Dropcam |

The experiment is repeated by giving the IoT devices a single label, to discriminate between traditional and IoT traffic using binary classification. The results for the Australia dataset are shown in Table 7.15. The classification achieves an accuracy of 99.0%, proving the high performance that can be obtained by applying this method.

Table 7.15: IoT Australia flows vs. NON_IoT flows, binary classification

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 92750 | 728 | 99.2% | 1.4% | 99.3% | 99.2% | 97.7% |
| NON_IoT | 45022 | 660 | 98.6% | 0.8% | 98.4% | 98.6% | 97.7% |
| **Weighted Avg.** | **137772** | **1388** | **99.0%** | **1.2%** | **99.0%** | **99.0%** | **97.7%** |

The same experiments are conducted on the California dataset, with the results shown in Tables 7.16 7.17 and 7.18. Similarly to the previous experiments, the accuracy is lower in this case. Nevertheless, the classifier is still able to distinguish the non-IoT flows well, giving an overall accuracy of 96.15% when the classifier distinguishes also the individual devices. The improvement however is only apparent. In fact, clearly the accuracy has improved because of the non-IoT flows, while the recognition of the devices has obviously worsen. In particular, several IoT flows are classified as non-IoT, producing a large increase of the False Positive rate of the non-IoT classThis is highlighted by the value of the MCC, whose weighted average (excluding the devices for which the computation does not provide a result) at 68.9% is much lower than in the case of the Australia dataset.

Table 7.16: California IoT with non-IoT traffic

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| HP printer | 0 | 4 | 0.0% | 0.0% | - | 0.0% | - |
| Amazon_Dash_Bounty_Button | 8 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo | 539 | 100 | 84.4% | 0.3% | 80.6% | 84.4% | 82.2% |
| Amazon_FireTVStick | 84 | 312 | 21.2% | 0.0% | 82.4% | 21.2% | 41.6% |
| AMCREST_IPCAM | 120 | 217 | 35.6% | 0.1% | 67.8% | 35.6% | 48.9% |
| Belkin_Wemo_Switch | 21 | 24 | 46.7% | 0.0% | 95.5% | 46.7% | 66.7% |
| D-link_IPCAM | 255 | 76 | 77.0% | 0.1% | 84.2% | 77.0% | 80.4% |
| FOSCAM_IPCAM | 421 | 614 | 40.7% | 0.3% | 72.8% | 40.7% | 53.7% |
| FOSCAM_IPCAM_vers2 | 42 | 45 | 48.3% | 0.0% | 72.4% | 48.3% | 59.1% |
| Google_Smartspeaker | 99 | 86 | 53.5% | 0.0% | 99.0% | 53.5% | 72.7% |
| Philips_Hue | 231 | 30 | 88.5% | 0.1% | 84.0% | 88.5% | 86.1% |
| RENPHO_humidifier | 0 | 5 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug | 30 | 9 | 76.9% | 0.0% | 88.2% | 76.9% | 82.4% |
| TP-Link SmartLightBulb | 1 | 3 | 25.0% | 0.0% | 50.0% | 25.0% | 35.4% |
| Wyze_IPCAM | 16 | 28 | 36.4% | 0.0% | 94.1% | 36.4% | 58.5% |
| NON_IoT | 45346 | 336 | 99.3% | 41.3% | 97.0% | 99.3% | 69.2% |
| **Weighted Avg.** | **47213** | **1890** | **96.2%** | **38.4%** | **95.8%** | **96.2%** | **68.9%** |

Table 7.17: Confusion matrix california IoT with non-IoT traffic

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | Classified as |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 45346 | 0 | 0 | 17 | 10 | 50 | 1 | 44 | 155 | 13 | 0 | 41 | 0 | 0 | 3 | 1 | 1 | a = NON_IOT |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | b = HP Printer california |
| 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | c = Amazon_Dash_Bounty_Button |
| 99 | 0 | 0 | 539 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d = Amazon Echo California |
| 196 | 0 | 0 | 106 | 84 | 4 | 0 | 2 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | e = Amazon_Fire_SmartTVStick |
| 213 | 0 | 0 | 0 | 1 | 120 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | f = AMCREST_IP_CAM |
| 21 | 0 | 0 | 0 | 2 | 0 | 21 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | g = Belkin Wemo switch california |
| 72 | 0 | 0 | 0 | 1 | 1 | 0 | 255 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | h = D-Link_IP_CAM |
| 612 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 421 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | i = Foscam_IP_CAM |
| 38 | 0 | 0 | 4 | 3 | 0 | 0 | 0 | 0 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | j = Foscam_IPCam2 |
| 86 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 99 | 0 | 0 | 0 | 0 | 0 | 0 | k = Google_SmartSpeaker |
| 28 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 231 | 0 | 0 | 0 | 0 | 0 | l = Philips-Hue |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | m = RENPHO_Humidifier |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | n = TENVIS_IPCam |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | o = TP-Link Smart plug california |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | p = TPLink_SmartLightBulb california |
| 26 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | q = Wyze_IPCam |

The binary classification results for the California dataset are shown in Table 7.18. Again, while accuracy is high, the MCC is much lower at 70.6% due to the high False Positive rate.

Table 7.18: IoT California flows vs. NON_IoT flows, binary classification

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 2124 | 1297 | 62.1% | 0.9% | 84.2% | 62.1% | 70.6% |
| NON_IoT | 45284 | 398 | 99.1% | 37.9% | 97.2% | 99.1% | 70.6% |
| **Weighted Avg.** | **47408** | **1695** | **96.5%** | **35.3%** | **96.3%** | **96.5%** | **70.6%** |

Our final results considering the entire dataset, including the simulated flows, are shown in Table 7.19 and 7.20. The accuracy for the recognition of the individual flows reaches 97.37% with an MCC of 96.5%. This can be considered a good result, considering the large number of flows in the dataset.

The results of binary classification with the entire dataset is finally shown in Table 7.20, with an accuracy of 98.0% and an overall MCC of 95.3%, highlighting again the reliability of the classification algorithm.

Table 7.19: Combined IoT flows with NON_IoT traffic

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| Dropcam | 198 | 58 | 77.3% | 0.0% | 97.5% | 77.3% | 86.8% |
| Smart Things | 26 | 6 | 81.3% | 0.0% | 100% | 81.3% | 90.1% |
| Withings Smart baby monitor | 4303 | 30 | 99.3% | 0.2% | 94.9% | 99.3% | 97.0% |
| Belkin Wemo Motion Sensor | 62392 | 48 | 99.9% | 0.7% | 99.0% | 99.9% | 99.1% |
| Samsung SmartCam | 5350 | 18 | 99.7% | 0.0% | 99.1% | 99.7% | 99.4% |
| Belkin_Wemo_Switch | 6248 | 40 | 99.4% | 0.0% | 99.6% | 99.4% | 99.5% |
| PIX-STAR Photo frame | 792 | 26 | 96.8% | 0.0% | 99.2% | 96.8% | 98.0% |
| Amazon_Echo | 2905 | 190 | 93.9% | 0.1% | 96.2% | 93.9% | 94.9% |
| TP-Link Smart Plug | 155 | 12 | 92.8% | 0.0% | 100.0% | 92.8% | 96.3% |
| Netatmo weather station | 1649 | 10 | 99.4% | 0.0% | 100.0% | 99.4% | 99.7% |
| TP-Link DayNight CloudCam | 801 | 50 | 94.1% | 0.0% | 98.9% | 94.1% | 96.5% |
| Netatmo Welcome | 1773 | 36 | 98.0% | 0.0% | 99.6% | 98.0% | 98.8% |
| Withings Smart Scale | 25 | 3 | 89.3% | 0.0% | 100% | 89.3% | 94.5% |
| Triby Speaker | 91 | 46 | 66.4% | 0.0% | 91.0% | 66.4% | 77.7% |
| NEST Protect smoke alarm | 51 | 16 | 76.1% | 0.0% | 98.1% | 76.1% | 86.4% |
| HP printer | 48 | 15 | 76.2% | 0.0% | 100.0% | 76.2% | 87.3% |
| Insteon Camera | 2998 | 0 | 100% | 0.0% | 99.7% | 100% | 99.9% |
| Withings AuraSmartSleepSensor | 1859 | 673 | 73.4% | 0.1% | 90.8% | 73.4% | 81.4% |
| iHome | 132 | 17 | 88.6% | 0.0% | 99.2% | 88.6% | 93.8% |
| Light Bulbs LiFX SmartBulb | 0 | 29 | 0.0% | 0.0% | - | 0.0% | - |
| Nest Dropcam | 258 | 101 | 71.9% | 0.0% | 98.1% | 71.9% | 83.9% |
| MIMIC | 6941 | 5 | 99.9% | 0.0% | 99.9% | 99.9% | 99.9% |
| HP printer Californian | 1 | 3 | 25.0% | 0.0% | 100.0% | 25.0% | 50.0% |
| Amazon_Dash_Bounty_Button | 8 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo Californian | 540 | 99 | 84.5% | 0.1% | 77.5% | 84.5% | 80.8% |
| Amazon_FireTVStick | 82 | 314 | 20.7% | 0.0% | 82.0% | 20.7% | 41.1% |
| AMCREST_IPCAM | 123 | 214 | 36.5% | 0.0% | 67.2% | 36.5% | 49.4% |
| Belkin_Wemo_Switch_Californian | 21 | 24 | 46.7% | 0.0% | 95.5% | 46.7% | 66.7% |
| D-link_IPCAM | 259 | 72 | 78.2% | 0.0% | 83.0% | 78.2% | 80.6% |
| FOSCAM_IPCAM | 413 | 622 | 39.9% | 0.1% | 72.0% | 39.9% | 53.4% |
| FOSCAM_IPCAM_vers2 | 38 | 49 | 43.7% | 0.0% | 67.9% | 43.7% | 54.4% |
| Google_Smartspeaker | 95 | 90 | 51.4% | 0.0% | 96.9% | 51.4% | 70.5% |
| Philips_Hue | 225 | 36 | 86.2% | 0.0% | 84.9% | 86.2% | 85.5% |
| RENPHO_humidifier | 0 | 5 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug Californian | 31 | 8 | 79.5% | 0.0% | 83.8% | 79.5% | 81.6% |
| TP-Link SmartLightBulb | 0 | 4 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| Wyze_IPCAM | 18 | 26 | 40.9% | 0.0% | 94.7% | 40.9% | 62.2% |
| NON_IoT | 44719 | 963 | 97.9% | 2.1% | 95.4% | 97.9% | 95.1% |
| **Weighted Avg.** | **145568** | **3959** | **97.4%** | **0.9%** | **97.2%** | **97.4%** | **96.5%** |

Table 7.20: Combined flows vs. NON_IoT flows, binary classification

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 102077 | 1768 | 98.3% | 2.7% | 98.8% | 98.3% | 95.3% |
| NON_IoT | 44448 | 1234 | 97.3% | 1.7% | 96.2% | 97.3% | 95.3% |
| **Weighted Avg.** | **146525** | **3002** | **98.0%** | **2.4%** | **98.0** | **98.0%** | **95.3%** |

### 7.3.4 Test-set evaluation

The above results were obtained through 10-fold cross validation. Because we leave the parameters of the models unchanged, this effectively partitions the dataset into a training and a test set, averaging the results across the different folds. In the following section, we will use the cross-validation technique to

tune the depth of the trees. In this case, we must set aside part of the dataset as a proper test set, which is not used during the tuning operation, to avoid overestimating the classifier performance. We opt for a 70%-30% split between training (used also in cross-validation) and test set, chosen randomly before applying the entire procedure. For sanity check, we have run the training procedure and test evaluation separately, without 10-fold cross validation, to verify that we obtain results that are consistent with those reported in the previous section. In tables 7.21 7.22 7.23 7.24 7.25 7.26 7.27 7.28 7.29 we see the new results. The results are similar to those of the 10-fold cross validation.

Table 7.21: Only australian IoT flows , classification, 30% test set

| **Device** | **Correct** | **Wrong** | **TP rate** | **FP rate** | **Precision** | **Recall** | **MCC** |
|---|---|---|---|---|---|---|---|
| Dropcam | 55 | 12 | 82.1% | 0.0% | 84.6% | 82.1% | 83.3% |
| Smart Things | 9 | 6 | 60.0% | 0.0% | 100% | 60.0% | 77.5% |
| Withings Smart baby monitor | 1212 | 1 | 99.9% | 0.0% | 99.8% | 99.9% | 99.9% |
| Belkin Wemo Motion Sensor | 18830 | 23 | 99.9% | 1.5% | 99.3% | 99.9% | 98.7% |
| Samsung SmartCam | 1582 | 5 | 99.7% | 0.0% | 99.3% | 99.7% | 99.5% |
| Belkin_Wemo_Switch | 1867 | 11 | 99.4% | 0.0% | 99.2% | 99.4% | 99.3% |
| PIX-STAR Photo frame | 242 | 2 | 99.2% | 0.0% | 98.4% | 99.2% | 98.8% |
| Amazon_Echo | 3028 | 871 | 97.5% | 0.1% | 96.8% | 97.5% | 97.1% |
| TP-Link Smart Plug | 44 | 0 | 100.0% | 0.0% | 97.8% | 100.0% | 98.9% |
| Netatmo weather station | 531 | 1 | 99.8% | 0.0% | 99.1% | 99.8% | 99.4% |
| TP-Link DayNight CloudCam | 264 | 8 | 97.1% | 0.0% | 99.6% | 97.1% | 98.3% |
| Netatmo Welcome | 542 | 3 | 99.4% | 0.0% | 98.9% | 99.4% | 99.2% |
| Withings Smart Scale | 5 | 1 | 83.3% | 0.0% | 100% | 83.3% | 91.3% |
| Triby Speaker | 43 | 6 | 87.8% | 0.0% | 97.7% | 87.8% | 92.6% |
| NEST Protect smoke alarm | 17 | 1 | 94.4% | 0.0% | 100% | 94.4% | 97.2% |
| HP printer | 24 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Insteon Camera | 864 | 0 | 100% | 0.0% | 99.7% | 100% | 99.8% |
| Withings AuraSmartSleepSensor | 650 | 130 | 83.3% | 0.0% | 99.1% | 83.3% | 90.6% |
| iHome | 50 | 3 | 94.3% | 0.0% | 100.0% | 94.3% | 97.1% |
| Light Bulbs LiFX SmartBulb | 3 | 1 | 75.0% | 0.0% | 60.0% | 75.0% | 67.1% |
| Nest Dropcam | 6 | 98 | 94.2% | 0.0% | 96.1% | 94.2% | 95.1% |
| **Weighted Avg.** | **27803** | **242** | **99.1%** | **1.1%** | **99.1%** | **99.1%** | **98.5%** |

Table 7.22: Only australian IoT flows vs. NON_IoT flows, classification, 30% test set

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| Dropcam | 65 | 25 | 72.2% | 0.0% | 94.2% | 72.2% | 82.5% |
| Smart Things | 7 | 3 | 70.0% | 0.0% | 100% | 70.0% | 83.7% |
| Withings Smart baby monitor | 1268 | 5 | 99.6% | 0.2% | 94.6% | 99.6% | 97.0% |
| Belkin Wemo Motion Sensor | 18558 | 13 | 99.9% | 0.8% | 99.1% | 99.9% | 99.1% |
| Samsung SmartCam | 1671 | 3 | 99.8% | 0.0% | 99.2% | 99.8% | 99.5% |
| Belkin_Wemo_Switch | 1928 | 15 | 99.2% | 0.0% | 99.8% | 99.2% | 99.5% |
| PIX-STAR Photo frame | 238 | 11 | 95.6% | 0.0% | 99.6% | 95.6% | 97.5% |
| Amazon_Echo | 844 | 43 | 94.2% | 0.1% | 96.7% | 94.2% | 95.3% |
| TP-Link Smart Plug | 40 | 6 | 87.0% | 0.0% | 100.0% | 87.0% | 93.2% |
| Netatmo weather station | 508 | 6 | 98.8% | 0.0% | 100.0% | 98.8% | 99.4% |
| TP-Link DayNight CloudCam | 258 | 19 | 93.1% | 0.0% | 98.9% | 93.1% | 95.9% |
| Netatmo Welcome | 523 | 14 | 97.4% | 0.0% | 99.6% | 97.4% | 98.5% |
| Withings Smart Scale | 5 | 1 | 83.3% | 0.0% | 100% | 83.3% | 91.3% |
| Triby Speaker | 26 | 17 | 60.5% | 0.0% | 96.3% | 60.5% | 76.3% |
| NEST Protect smoke alarm | 12 | 5 | 70.6% | 0.0% | 100% | 70.6% | 84.0% |
| HP printer | 18 | 3 | 85.7% | 0.0% | 100.0% | 85.7% | 92.6% |
| Insteon Camera | 924 | 0 | 100% | 0.0% | 99.8% | 100% | 99.9% |
| Withings AuraSmartSleepSensor | 559 | 195 | 74.1% | 0.1% | 91.8% | 74.1% | 82.2% |
| iHome | 41 | 5 | 89.1% | 0.0% | 100.0% | 89.1% | 94.4% |
| Light Bulbs LiFX SmartBulb | 0 | 12 | 0.0% | 0.0% | - | 0.0% | - |
| Nest Dropcam | 66 | 39 | 62.9% | 0.0% | 98.5% | 62.9% | 78.6% |
| NON_IoT | 45087 | 595 | 98.7% | 1.0% | 98.0% | 98.7% | 97.5% |
| **Weighted Avg.** | **41114** | **629** | **98.5%** | **0.7%** | **98.4%** | **98.5%** | **98.0%** |

Table 7.23: Only australian IoT flows vs. NON_IoT flows, binary classification, 30% test set

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 27787 | 221 | 99.2% | 1.5% | 99.3% | 99.2% | 97.7% |
| NON_IoT | 13529 | 206 | 98.5% | 0.8% | 98.4% | 98.5% | 97.7% |
| **Weighted Avg.** | **41316** | **427** | **99.0%** | **1.3%** | **99.0%** | **99.0%** | **97.7%** |

Table 7.24: Only californian IoT, classification, 30% test set

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| HP printer | 0 | 0 | 0.0% | 0.0% | - | 0.0% | - |
| Amazon_Dash_Bounty_Button | 2 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo | 181 | 7 | 96.3% | 7.5% | 74.2% | 96.3% | 83.8% |
| Amazon_FireTVStick | 48 | 62 | 43.5% | 2.1% | 71.6% | 43.6% | 52.9% |
| AMCREST_IPCAM | 87 | 19 | 82.1% | 1.7% | 84.5% | 82.1% | 81.4% |
| Belkin_Wemo_Switch | 12 | 3 | 80.0% | 0.3% | 80.0% | 80.0% | 79.7% |
| D-link_IPCAM | 87 | 10 | 89.7% | 0.8% | 92.6% | 89.7% | 90.2% |
| FOSCAM_IPCAM | 285 | 16 | 94.7% | 2.2% | 94.7% | 94.7% | 94.7% |
| FOSCAM_IPCAM_vers2 | 21 | 12 | 63.6% | 0.3% | 87.5% | 63.6% | 73.9% |
| Google_Smartspeaker | 53 | 2 | 96.4% | 1.2% | 81.5% | 96.4% | 88.0% |
| Philips_Hue | 79 | 8 | 90.8% | 1.3% | 86.8% | 90.8% | 87.7% |
| RENPHO_humidifier | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 0 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug | 8 | 3 | 72.7% | 0.1% | 88.9% | 72.7% | 80.2% |
| TP-Link SmartLightBulb | 0 | 1 | 0.0% | 0.1% | 0.0% | 0.0% | 0.0% |
| Wyze_IPCAM | 8 | 8 | 50.0% | 0.1% | 88.9% | 50.0% | 66.3% |
| **Weighted Avg.** | **871** | **154** | **85.0%** | **2.7%** | **84.68** | **83.0%** | **83.66** |

Table 7.25: Only californian IoT flows vs. NON_IoT flows, classification, 30% test set

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| HP printer | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| Amazon_Dash_Bounty_Button | 4 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo | 164 | 27 | 85.9% | 0.3% | 78.8% | 85.9% | 82.0% |
| Amazon_FireTVStick | 19 | 94 | 14.4% | 0.0% | 90.5% | 14.4% | 35.9% |
| AMCREST_IPCAM | 33 | 81 | 28.9% | 0.1% | 64.7% | 28.9% | 43.0% |
| Belkin_Wemo_Switch | 6 | 6 | 50.0% | 0.0% | 85.7% | 50.0% | 65.4% |
| D-link_IPCAM | 75 | 16 | 82.4% | 0.1% | 84.3% | 82.4% | 83.2% |
| FOSCAM_IPCAM | 137 | 158 | 46.4% | 0.3% | 73.3% | 46.4% | 57.7% |
| FOSCAM_IPCAM_vers2 | 16 | 16 | 50.0% | 0.0% | 69.6% | 50.0% | 58.9% |
| Google_Smartspeaker | 30 | 28 | 51.7% | 0.0% | 100.0% | 51.7% | 71.9% |
| Philips_Hue | 66 | 12 | 84.6% | 0.1% | 86.6% | 84.6% | 85.6% |
| RENPHO_humidifier | 0 | 3 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug | 3 | 4 | 57.1% | 0.0% | 66.7% | 57.1% | 61.7% |
| TP-Link SmartLightBulb | 1 | 3 | - | - | - | - | - |
| Wyze_IPCAM | 4 | 8 | 33.3% | 0.0% | 100.0% | 33.3% | 57.7% |
| NON_IoT | 13604 | 107 | 99.3% | 41.1% | 97.0% | 99.3% | 69.4% |
| **Weighted Avg.** | **14162** | **573** | **96.1%** | **38.3%** | **95.7%** | **96.1%** | **69.0%** |

Table 7.26: Only Californian IoT flows vs. NON_IoT flows, binary classification, 30% test set

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 644 | 387 | 62.5% | 0.9% | 84.1% | 62.5% | 70.8% |
| NON_IoT | 13582 | 122 | 99.1% | 37.5% | 97.2% | 99.1% | 70.8% |
| **Weighted Avg.** | **14226** | **509** | **96.5%** | **35.0%** | **96.3%** | **96.5%** | **70.8%** |

Table 7.27: Combined IoT flows, classification, 30% test

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| Dropcam | 65 | 10 | 86.7% | 0.0% | 84.4% | 86.7% | 85.5% |
| Smart Things | 5 | 3 | 62.5% | 0.0% | 100.0% | 62.5% | 79.1% |
| Withings Smart baby monitor | 1287 | 1 | 99.9% | 0.0% | 99.5% | 99.9% | 99.7% |
| Belkin Wemo Motion Sensor | 18720 | 10 | 99.9% | 1.2% | 99.2% | 99.9% | 98.9% |
| Samsung SmartCam | 1631 | 2 | 99.9% | 0.0% | 99.5% | 99.9% | 99.6% |
| Belkin_Wemo_Switch | 1923 | 6 | 99.7% | 0.0% | 99.6% | 99.7% | 99.6% |
| PIX-STAR Photo frame | 227 | 3 | 98.7% | 0.0% | 98.7% | 98.7% | 98.7% |
| Amazon_Echo | 899 | 20 | 97.8% | 0.2% | 94.5% | 97.8% | 96.0% |
| TP-Link Smart Plug | 41 | 1 | 97.6% | 0.0% | 95.3% | 97.6% | 96.5% |
| Netatmo weather station | 531 | 2 | 99.6% | 0.0% | 99.4% | 99.6% | 99.5% |
| TP-Link DayNight CloudCam | 235 | 16 | 93.6% | 0.0% | 98.7% | 93.6% | 96.1% |
| Netatmo Welcome | 506 | 5 | 99.0% | 0.1% | 96.7% | 99.0% | 97.8% |
| Withings Smart Scale | 7 | 0 | 100.0% | 0.0% | 100% | 100.0% | 100.0% |
| Triby Speaker | 34 | 3 | 91.9% | 0.0% | 89.5% | 91.9% | 90.7% |
| NEST Protect smoke alarm | 15 | 1 | 93.8% | 0.0% | 100.0% | 93.8% | 96.8% |
| HP printer | 14 | 2 | 87.5% | 0.0% | 77.8% | 87.5% | 82.5% |
| Insteon Camera | 878 | 0 | 100% | 0.0% | 99.8% | 100% | 99.9% |
| Withings AuraSmartSleepSensor | 623 | 140 | 81.7% | 0.1% | 95.7% | 81.7% | 88.1% |
| iHome | 48 | 6 | 88.9% | 0.0% | 94.1% | 88.9% | 91.5% |
| Light Bulbs LiFX SmartBulb | 4 | 2 | 66.7% | 0.0% | 100.0% | 66.7% | 81.6% |
| Nest Dropcam | 107 | 13 | 89.2% | 0.0% | 88.4% | 89.2% | 88.8% |
| MIMIC | 2049 | 1 | 100% | 0.0% | 99.5% | 100.0% | 99.7% |
| HP printer Californian | 0 | 0 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| Amazon_Dash_Bounty_Button | 0 | 4 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| Amazon_Echo Californian | 187 | 12 | 94.0% | 0.2% | 70.8% | 94.0% | 81.5% |
| Amazon_FireTVStick | 42 | 88 | 32.6% | 0.0% | 80.8% | 32.6% | 51.2% |
| AMCREST_IPCAM | 76 | 30 | 71.7% | 0.0% | 84.4% | 71.7% | 77.7% |
| Belkin_Wemo_Switch_Californian | 8 | 3 | 72.7% | 0.0% | 80.0% | 72.7% | 76.3% |
| D-link_IPCAM | 91 | 12 | 88.3% | 0.0% | 91.0% | 88.3% | 89.6% |
| FOSCAM_IPCAM | 294 | 27 | 91.6% | 0.0% | 97.4% | 91.6% | 94.4% |
| FOSCAM_IPCAM_vers2 | 21 | 11 | 65.6% | 0.0% | 77.8% | 65.6% | 71.4% |
| Google_Smartspeaker | 41 | 16 | 71.9% | 0.0% | 78.8% | 71.9% | 75.3% |
| Philips_Hue | 71 | 10 | 87.7% | 0.0% | 98.6% | 87.7% | 93.0% |
| RENPHO_humidifier | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 0 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug Californian | 4 | 1 | 80.0% | 0.0% | 66.7% | 80.0% | 73.0% |
| TP-Link SmartLightBulb | 0 | 1 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| Wyze_IPCAM | 8 | 8 | 50.0% | 0.0% | 100.0% | 50.0% | 70.7% |
| **Weighted Avg.** | **30692** | **471** | **98.5%** | **0.7%** | **98.3%** | **98.5%** | **97.9%** |

Table 7.28: Combined flows vs. NON_IoT flows, classification, 30% test

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| Dropcam | 62 | 13 | 82.7% | 0.0% | 98.4% | 82.7% | 90.2% |
| Smart Things | 7 | 2 | 77.8% | 0.0% | 100% | 77.8% | 88.2% |
| Withings Smart baby monitor | 1295 | 8 | 99.3% | 0.2% | 94.3% | 99.3% | 96.6% |
| Belkin Wemo Motion Sensor | 18738 | 12 | 99.9% | 0.7% | 99.0% | 99.9% | 99.1% |
| Samsung SmartCam | 1574 | 4 | 99.7% | 0.0% | 99.2% | 99.7% | 99.5% |
| Belkin_Wemo_Switch | 1848 | 14 | 99.2% | 0.0% | 99.9% | 99.2% | 99.5% |
| PIX-STAR Photo frame | 229 | 11 | 95.4% | 0.0% | 99.1% | 95.4% | 97.2% |
| Amazon_Echo | 860 | 48 | 94.5% | 0.1% | 95.6% | 94.5% | 94.9% |
| TP-Link Smart Plug | 57 | 3 | 95.0% | 0.0% | 100.0% | 95.0% | 97.5% |
| Netatmo weather station | 504 | 2 | 99.6% | 0.0% | 100.0% | 99.6% | 99.8% |
| TP-Link DayNight CloudCam | 229 | 18 | 92.7% | 0.0% | 98.3% | 92.7% | 95.4% |
| Netatmo Welcome | 525 | 19 | 96.5% | 0.0% | 99.4% | 96.5% | 97.9% |
| Withings Smart Scale | 13 | 1 | 92.9% | 0.0% | 100% | 92.9% | 96.4% |
| Triby Speaker | 21 | 17 | 55.3% | 0.0% | 95.5% | 55.3% | 72.6% |
| NEST Protect smoke alarm | 9 | 4 | 69.2% | 0.0% | 100.0% | 69.2% | 83.2% |
| HP printer | 23 | 8 | 74.2% | 0.0% | 100.0% | 74.2% | 86.1% |
| Insteon Camera | 914 | 0 | 100% | 0.0% | 99.7% | 100% | 99.8% |
| Withings AuraSmartSleepSensor | 555 | 200 | 73.5% | 0.1% | 89.7% | 73.5% | 80.9% |
| iHome | 41 | 3 | 93.2% | 0.0% | 100.0% | 93.2% | 96.5% |
| Light Bulbs LiFX SmartBulb | 0 | 10 | 0.0% | 0.0% | - | 0.0% | - |
| Nest Dropcam | 77 | 48 | 61.6% | 0.0% | 93.6% | 61.6% | 77.0% |
| MIMIC | 2128 | 3 | 99.9% | 0.0% | 100.0% | 99.9% | 99.9% |
| HP printer Californian | 0 | 0 | - | - | - | - | - |
| Amazon_Dash_Bounty_Button | 2 | 0 | 100.0% | 0.0% | 100.0% | 100.0% | 100.0% |
| Amazon_Echo Californian | 161 | 37 | 81.3% | 0.1% | 77.0% | 81.3% | 79.0% |
| Amazon_FireTVStick | 20 | 68 | 18.5% | 0.0% | 80.0% | 18.5% | 38.4% |
| AMCREST_IPCAM | 34 | 65 | 36.5% | 0.0% | 67.2% | 36.5% | 49.4% |
| Belkin_Wemo_Switch_Californian | 8 | 11 | 42.1% | 0.0% | 80.0% | 42.1% | 58.0% |
| D-link_IPCAM | 80 | 26 | 75.5% | 0.0% | 86.0% | 75.5% | 80.5% |
| FOSCAM_IPCAM | 127 | 173 | 42.3% | 0.1% | 73.4% | 42.3% | 55.5% |
| FOSCAM_IPCAM_vers2 | 13 | 13 | 50.0% | 0.0% | 65.0% | 50.0% | 57.0% |
| Google_Smartspeaker | 30 | 31 | 49.2% | 0.0% | 93.8% | 49.2% | 67.9% |
| Philips_Hue | 78 | 13 | 85.7% | 0.0% | 88.6% | 85.7% | 87.1% |
| RENPHO_humidifier | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TENVIS_IPCAM | 0 | 1 | 0.0% | 0.0% | - | 0.0% | - |
| TP-Link Smart Plug Californian | 8 | 3 | 72.7% | 0.0% | 72.7% | 72.7% | 72.7% |
| TP-Link SmartLightBulb | 0 | 1 | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| Wyze_IPCAM | 6 | 8 | 40.0% | 0.0% | 85.7% | 40.0% | 58.5% |
| NON_IoT | 13379 | 281 | 97.9% | 2.1% | 95.3% | 97.9% | 95.1% |
| **Weighted Avg.** | **30692** | **471** | **98.5%** | **0.7%** | **97.0%** | **98.5%** | **96.5** |

Table 7.29: Combined flows vs. NON_IoT flows, binary classification, 30% test set

| Device | Correct | Wrong | TP rate | FP rate | Precision | Recall | MCC |
|---|---|---|---|---|---|---|---|
| IoT | 30659 | 530 | 98.3% | 2.8% | 98.8% | 98.3% | 95.2% |
| NON_IoT | 13290 | 381 | 97.2% | 1.7% | 96.2% | 97.2% | 95.2% |
| **Weighted Avg.** | **43949** | **911** | **98.0%** | **2.5%** | **98.0%** | **98.0%** | **95.2%** |

### 7.3.5 Hyper-parameter Tuning

The previous results were obtained using the default settings for training the Random Forest classifier. In this section we explore the impact of changing the maximum allowed depth of the trees. This could be useful for two reasons. First, a larger depth leads to models with higher variance, making the

classifier more prone to overfitting. Second, performance may tend to level off with tree depth. In that case, choosing the smallest depth that provides acceptable performance can greatly simplify the evaluation of the model, with lower computational complexity and lower latency.

As in the previous section, the analysis is conducted by splitting the dataset between 70% for training and 30% for test. We analyze the performance of the classifier by performing 5-fold cross validation on the training set (which is, therefore, automatically split between a proper training and a validation set), with tree depth limited to 1, 2, 5, 10, 20, 40 and 80 levels. While we record all the performance metrics, we report here only the MCC, as the other measures follow a similar pattern, and use it to select the optimal depth. The test set is then used to quantify the actual performance of the classifier for the selected depth.

Figure 7.13 shows the results for the IoT devices only, for the Australia, the California and the combined datasets, for tree depth up to 40 levels (the values for higher depths do not change relative to 40 levels, and are therefore not shown). The plot includes the results of cross-validation, as well as the results on the test set for all tree depths, where the test set results are significant only after the choice of depth (i.e., they are not used for tuning). In all cases, the performance levels off at a depth of 10 levels, and does not exhibit overfitting. The California dataset, as observed in previous sections, has the worst performance, and is considerably lower than the other cases at small tree depths. The performance of the classifier on the test set, here as in the subsequent experiments, does not change significantly relative to the cross validation.

Figure 7.14 shows the same data with the inclusion of the non-IoT flows, while Figure 7.15 illustrates the performance of the binary IoT/non-IoT classification. The non-IoT flows bring more variety and negatively affect the performance: slightly for the Australia dataset, more significantly for the California dataset. In this last case, tree depths less than 5 result in essentially a random classifier. The combined dataset has better performance, largely because of the contribution of the Australia dataset as discussed previously.

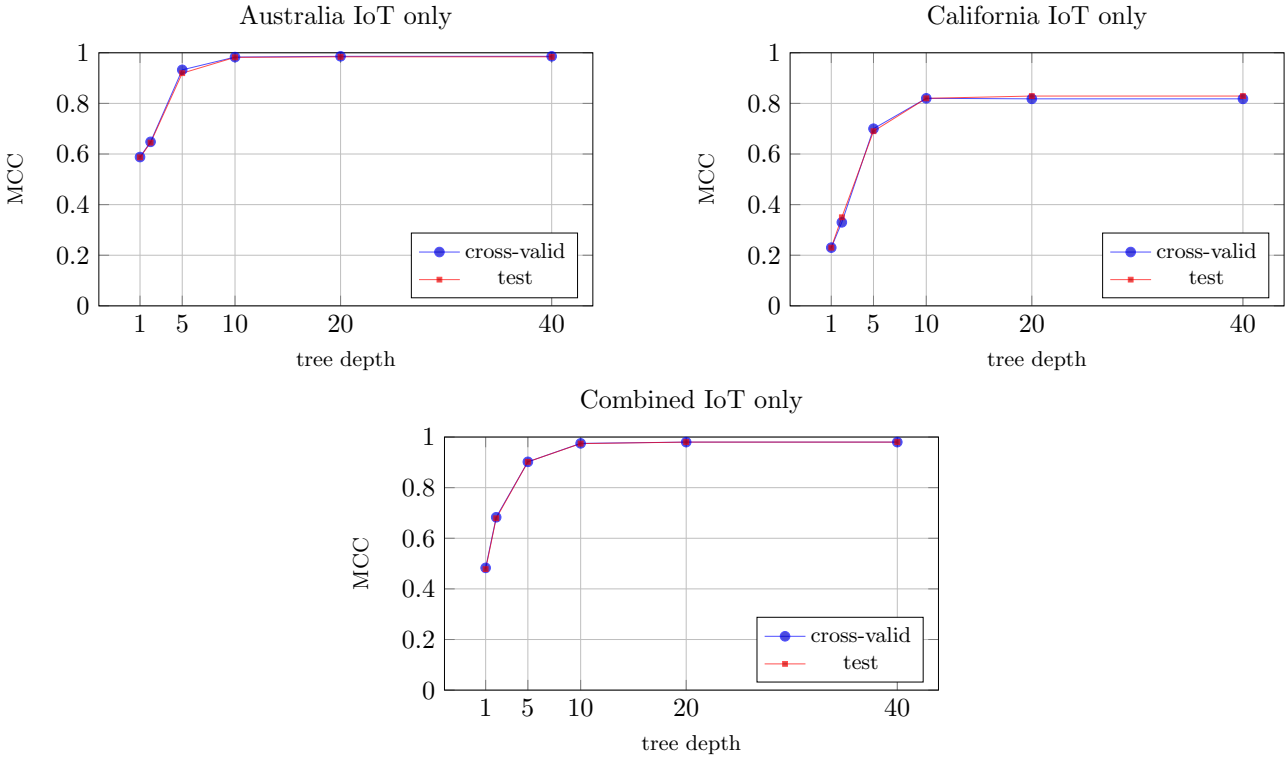



Figure 7.13: MCC of training and test set as the tree depth varies from 1 to 40 for IoT only

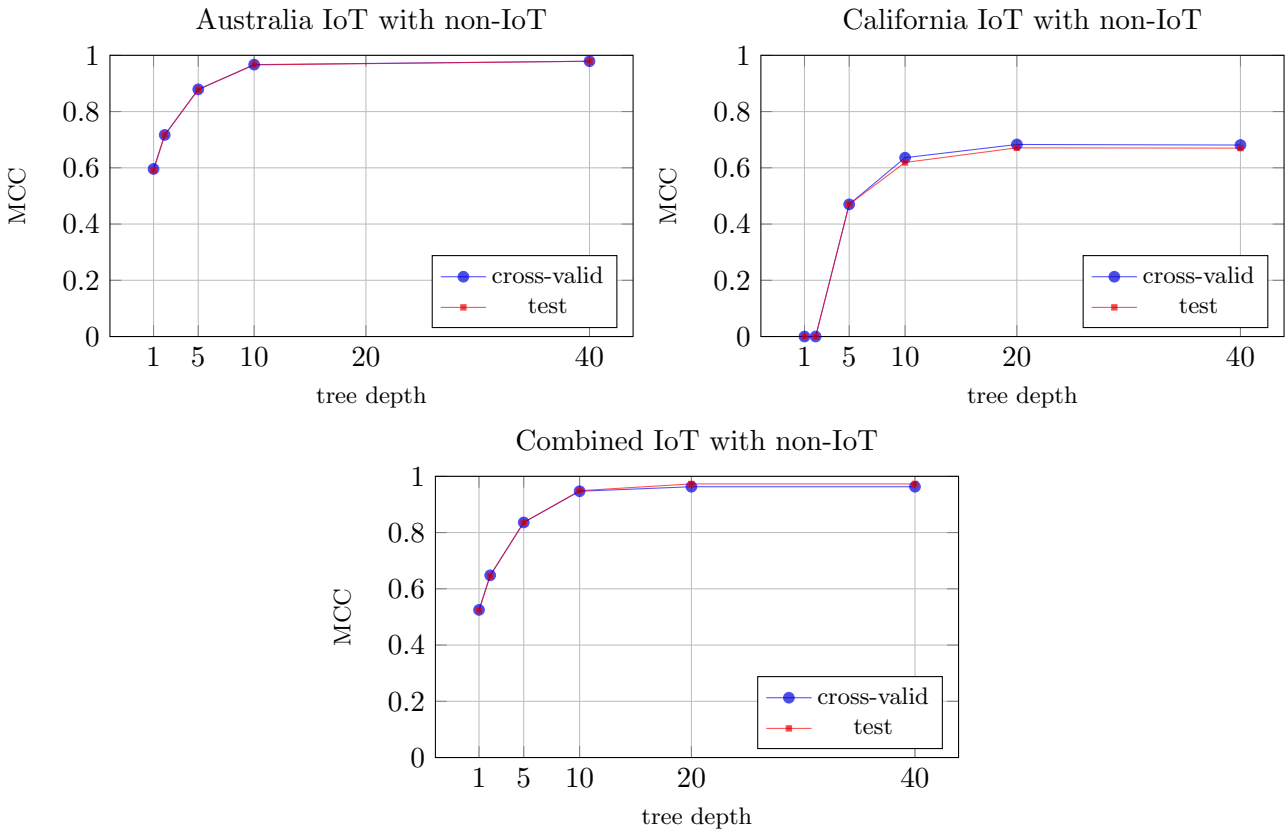



Figure 7.14: MCC of training and test set as the tree depth varies from 1 to 40 for IoT with non-IoT

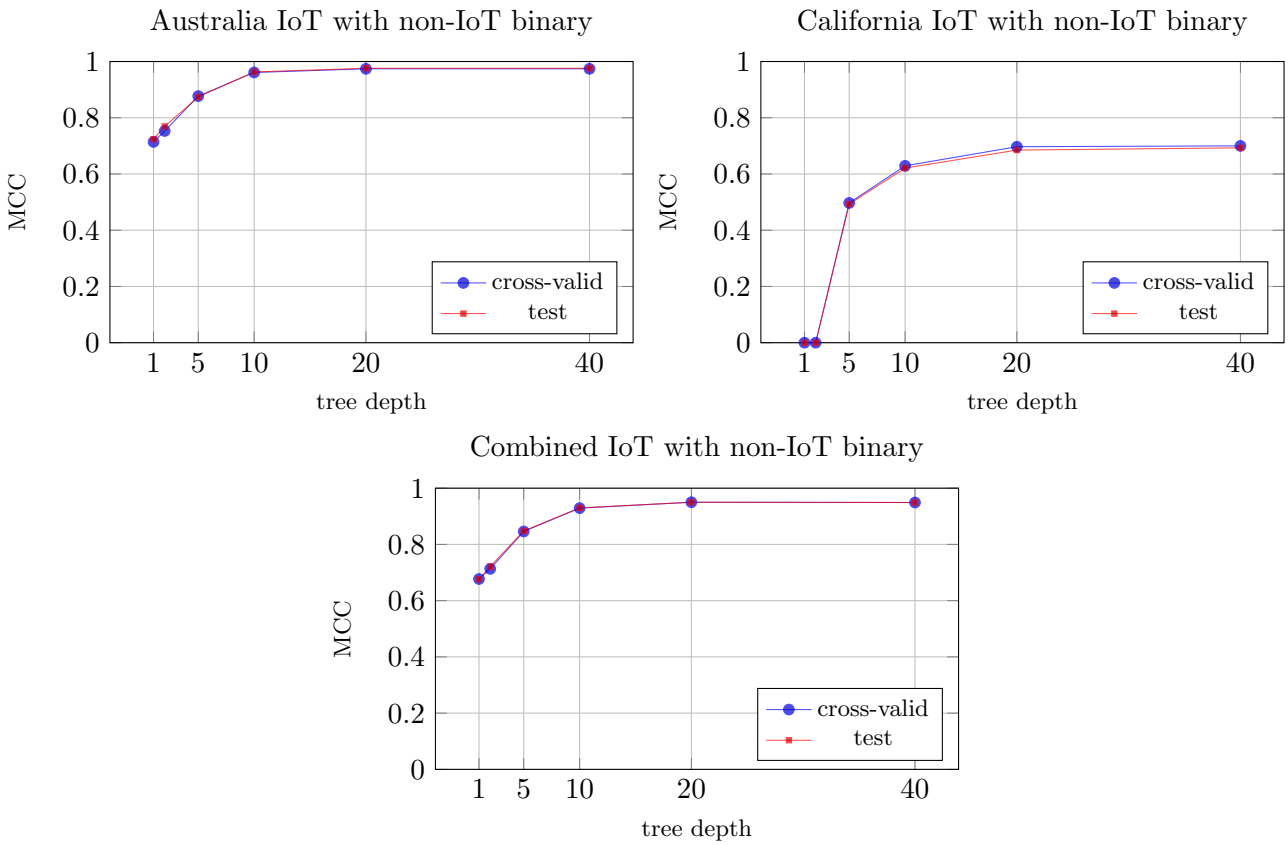



Figure 7.15: MCC of training and test set as the tree depth varies from 1 to 40 for IoT and non-IoT, binary classification

It is interesting to validate a classifier derived from one dataset on another dataset. The application is problematic since the devices are not generally the same, but for a few exceptions. We have therefore trained the classifier on the Australia dataset, and validated the model on the California dataset limited to the common devices, i.e., the Belkin Wemo switch, the Amazon Echo and the TP-Link Smart plug, which were given the same labels in the two datasets. The performance in terms of the MCC is shown in Figure 7.16.

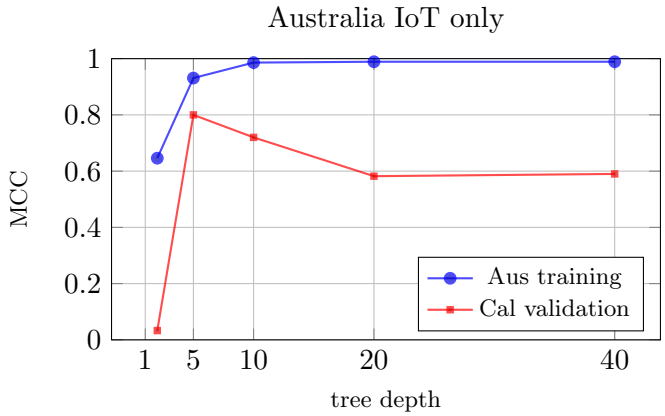



Figure 7.16: Classifier trained on the Australia dataset, validated on the California dataset

The results lead to two observations. First, the classifier is sensitive to the specific deployment, and does not perform as well on a different deployment, despite the use of the same devices. This may be due to different configurations of the devices, which potentially interact with different servers, altering the dynamics of the communication. More interestingly, we see that the classifier achieves the best generalization at a depth in the range of 5 to 10 levels, with the performance on the validation set decreasing for larger depths. This underscores some amount of variance when the model becomes too complex.

### 7.3.6 Testset with non_IoT

We tried to test our models with a testset composed of the NON_IoT flows collected in a domestic environment. We collected a total of 5587 test NON_IoT flows.
We compared this testset with the following models:

- IoT-non_IoT australian dataset only, devices
- IoT-non_IoT australian dataset only, binary
- IoT-non_IoT californian dataset only, devices
- IoT-non_IoT californian dataset only, binary
- IoT-non_IoT all datasets, devices
- IoT-non_IoT all datasets, binary
- IoT-non_IoT australian dataset only, devices

Table 7.30: IoT-non_IoT australian flows compared with testset flows devices

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 5454 - 97.62% | 133 - 2.38% | 5587 - 100% |

The confusion matrix is:

Table 7.31: IoT-non_IoT australian flows compared with testset flows devices

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 5454 | 133 | NON_IoT |
| 0 | 0 | IoT |

- IoT-non_IoT australian dataset only, binary

Table 7.32: IoT-non_IoT australian flows compared with testset flows binary

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 5110 - 91.46% | 477 - 8.54% | 5587 - 100% |

The confusion matrix is:

Table 7.33: IoT-non_IoT australian flows compared with testset flows binary

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 5110 | 477 | NON_IoT |
| 0 | 0 | IoT |

- IoT-non_IoT californian dataset only, devices

Table 7.34: IoT-non_IoT californian flows compared with testset flows devices

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 5270 - 94.33% | 317 - 5.67% | 5587 - 100% |

The confusion matrix is:

Table 7.35: IoT-non_IoT californian flows compared with testset flows devices

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 5270 | 317 | NON_IoT |
| 0 | 0 | IoT |

- IoT-non_IoT californian dataset only, binary

Table 7.36: IoT-non_IoT californian flows compared with testset flows binary

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 5105 - 91.37% | 482 - 8.63% | 5587 - 100% |

The confusion matrix is:

Table 7.37: IoT-non_IoT californian flows compared with testset flows binary

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 5105 | 482 | NON_IoT |
| 0 | 0 | IoT |

- IoT-non_IoT all dataset only,devices

Table 7.38: IoT-non_IoT all flows compared with testset flows devices

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 5104 - 91.35% | 483 - 8.65% | 5587 - 100% |

The confusion matrix is:

Table 7.39: IoT-non_IoT all flows compared with testset flows devices

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 5104 | 483 | NON_IoT |
| 0 | 0 | IoT |

- IoT-non_IoT all dataset only,binary

Table 7.40: IoT-non_IoT all flows compared with testset flows binary

| Correctly Classified Instances | InCorrectly Classified Instances | Total Number of Instances |
|---|---|---|
| 4207 - 75.30% | 1380 - 24.70% | 5587 - 100% |

The confusion matrix is:

Table 7.41: IoT-non_IoT all flows compared with testset flows binary

| **NON_IoT** | **IoT** | **Classified as** |
|---|---|---|
| 4207 | 1380 | NON_IoT |
| 0 | 0 | IoT |

### 7.3.7 Discussion

The classification results shown in the previous sections highlight that the analysis of the series of the payload lengths in the frequency domain provides high selectivity, with high overall accuracy and MCC. On the other hand, the analysis also shows that smaller datasets may be unable to provide sufficient training examples to reduce the classification error. In this case, the model may tend to overfit the data, and the validation suffers. One possible remedy in these cases is to further reduce the number of features. In a preliminary experiment, we have considered a reduced subset of the Australia dataset corresponding to only three days of data capture with approximately 17,000 flows. We have then isolated only a reduced number of peaks in the spectrum. The experiments show that using only 32, 16 or even 4 peaks reduces accuracy by only 0.2% points, showing that the highest values of the spectrum carry the majority of the information. A more detailed evaluation of this approach is part of our current and future work.

The classification accuracy and performance in terms of precision, recall and the other metrics that we obtain is in line with the results obtained in our previous work [19], however we here use a very different set of features. More specifically, we focus on simplified features based on the length of the packets (in some cases complemented by the inter-arrival times) and further evaluate the performance on a much larger dataset. This is convenient for both computational complexity, if the recognition must be run in real time, and to handle flows that are obfuscated by encryption. A combined classifier could also be used to increase the reliability. In addition, we have extended the classifier to distinguish between the different classes of devices, a valuable information to adjust the network quality of service, to detect anomalous behavior and therefore isolate compromised devices.

Other methods, discussed in Section 7.1, have also been presented in the literature that reach high classification accuracy, selectivity and specificity [47, 45, 40, 28]. Our intention was not so much improving the performance metrics, which arguably reach almost perfect classification in some of the reported work, but rather achieve similar performance using a considerably reduced set of features

that does not need header or payload inspection. By doing so, the classifier could be deployed also in the presence of encryption. In addition, we study the performance of classification in the frequency domain.

In our study we also explore the impact of tree depth in terms of classification performance. The results show that relatively shallow trees already provide the bulk of the distinguishing power of the method, while resulting in a much simpler implementation, essential when dealing with traffic in real time. We have also considered applying the classifier obtained with one dataset to an entirely different dataset. This, as far as we know, has never been attempted in the literature. The analysis, limited to the common devices, shows that i) performance suffers, and ii) deep trees may overfit relative to another dataset. The tree depth analysis identifies the optimal value for the hyper-parameter in this case. Nevertheless, the models show some difficulty in generalizing, presumably because of the wide differences in configurations and environment of operation. This aspect is largely unexplored in the state of the art, and is hampered by the lack of appropriate labeled datasets. One mention is given by Pinheiro et al. [40], who observe similar behavior with firmware updates or compromised devices, and suggest, without going further, that unsupervised learning techniques could be used to address the problem.

# 8 Conclusions

Deep packet inspection can be applied whenever access to the transport payload is granted to the application. However, encryption is increasingly adopted to protect the data. Depending on the level of encryption, DPI might be rendered ineffective. In particular, if mechanisms such as TLS are used, anything above the transport level is not accessible, leaving only address and port information visible. Other methods, such IPsec and Virtual Private Networks (VPN) make DPI completely unusable, as ports are invisible and traffic from different sources may be mixed in the same logical flow. Statistical and behavioral classification can help overcome these difficulties. More specifically, these methods are relatively insensitive to the application protocol in use. Hence, even generic HTTP connections could be detected and classified correctly. In addition, they can work in the presence of encryption, as long as the traffic maintains its peculiar characteristics. For example, a VPN that tunnels several flows could easily confuse the best statistical and behavioral classifier. The downside of these methods is that the system must store potentially substantial per-flow information, in order to build the necessary parameters for identification. In addition, the classifier must generally be trained appropriately, which requires generating or collecting representative labeled traffic. Shifts in traffic patterns may require retraining the classification network, an operation which is difficult to perform on the fly. The other disadvantage is that classification occurs some time after the beginning of the flow. This may be inappropriate for applications where packets must be routed immediately depending on their type. In the end, behavioral methods rely on distinctive communication patterns that help discriminate user traffic from machine generated traffic. As IoT devices become more "intelligent" and multi-function, however, their traffic patterns become correspondingly more diverse and less predictable. Statistical and behavioral classification are therefore expected to behave less accurately. Thus, in general, a combination of techniques is likely to provide the best results, especially in terms of performance. For instance, range-based classification and Bloom filters could be used for an initial screening of the packets. This allows the system to perform a more in depth analysis to only those packets or flows which are of more interest, or that defy a simpler mechanism. Regular expression can then be used for payload inspection, when possible, while statistical and behavioral classification would be dedicated to those flows that offer lower visibility. As line speeds increase, it is likely that a multiplicity of approaches will be employed, together with increasing parallelism. In this work, we have studied a statistical classification method to discriminate between IoT and non-IoT traffic, and to determine the device that originates the communication flow. We have first presented and characterized our dataset, collected from repositories made available in the public domain, discussed the tools we have used to capture the data, generate the flows and their statistics, and construct the classification algorithms. We have shown the features of different classes of devices, and discussed the classification performance of the J48 and Random Forest algorithms using 10-fold cross validation.

Our future work is moving in two directions. The first is to explore the timing relation among different flows attributed to the same device. The difficulty with this kind of analysis is that regularity is seen across the flows which are opened and closed by a particular device. A different criterion must therefore be employed. In the context of cellular networks, one can use a specific identifiers, like the IMSI, to associate traffic to a device. At the same time, the same device, such as a smartphone, might behave as both a "thing" (by using its sensors) and a traditional terminal, creating confusion or noise in the classification. Another direction includes a form of dynamic learning to follows the evolution of the behavior of devices, in a spirit similar to the self-learning classifier proposed by Grimaudo et al. [22]. In our case, we could, in principle, evaluate the degree of confidence in classification by looking at where in the tree the flows are classified. In fact, the tree identifies hypercubes in the attribute space which divide the IoT from the non-IoT flows. The distance from the hypercube edge could be considered a measure of confidence. Training could be performed at run-time, using flows classified with high confidence as training data. Additional information, such as the port number (as used

in [22]), could also be employed to estimate accuracy, when the information is available. When the system observes that the overall classification confidence has decreased significantly, a new round of supervised learning could be employed to restore the lost accuracy.

# Bibliography


[1] Contiki. http://www.contiki-os.org/.

[2] Ibm cloud. https://www.ibm.com/cloud/.

[3] Libpcap. https://www.tcpdump.org/manpages/pcap.3pcap.html.

[4] Mark borgerding. kissftt. https://github.com/mborgerding/kissfft.

[5] MIMIC MQTT simulator. https://www.gambitcomm.com/site/.

[6] Mqtt version 3.1.1, 2014. oasis standard.

[7] Node-red. https://nodered.org/.

[8] Sniffex. https://www.tcpdump.org/sniffex.c. Version 0.1.1 (2005-07-05) Copyright (c) 2005 The Tcpdump Group.

[9] Ian H.Witten Eibe Frank Mark A.Hall and Christopher J.Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, fourth edition edition, 2016.

[10] Michela Meo Alessandro Finamore, Marco Mellia and Dario Rossi. Kiss: Stochastic packet inspection classifier for udp traffic. In *IEEE/ACM Transactions on Networking*, 2010.

[11] A. Amouri, V.T. Alaparthy, and S.D. Morgera. A machine learning based intrusion detection system for mobile internet of things. *Sensors*, 20(2):461, 2020.

[12] AppNeta. Tcpreplay: Pcap editing and replaying utilities, 2015. `http://tcpreplay.appneta.com/`.

[13] A. Bär, P. Svoboda, and P. Casas. MTRAC - discovering M2M devices in cellular networks from coarse-grained measurements. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 667–672, London, UK, 8-12 June 2015.

[14] Gennaro Cirillo and Roberto Passerone. Packet length spectral analysis for iot flow classification using ensemble learning. *Special Issue Applications in Electronics Pervading Industry, Environment and Society – Sensing Systems and Pervasive Intelligence, Sensors Journal, Submitted for pubblication*, 2020.

[15] A. Dainotti, A. Pescapè, P. Salvo Rossi, F. Palmieri, and G. Ventre. Internet traffic modeling by means of hidden markov models. *Computer Networks*, 52(14):2645–2662, October 2008.

[16] A. Dainotti, A. Pescapè, and G. Ventre. A cascade architecture for DoS attacks detection based on the wavelet transform. *Journal of Computer Security*, 17(6/2009):945–968, 2009.

[17] Giovanni Vigna Christopher Kruegel Florian Tegeler, Xiaoming Fu. Botfinder: Finding bots in network traffic without deep packet inspection. *CoNEXT '12: Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012.

[18] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *ACM CoNEXT10*, Philadelphia, PA, December 2010.

[19] Antonio Posenato Gennaro Cirillo, Roberto Passerone and Luca Rizzon. Statistical flow classification for the iot. *Applications in Electronics Pervading Industry, Environment and Society, ApplePies 2019, Pisa, Italy, September 11–13, 2019.*, 2019.

[20] Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A. Ghorbani. Characterization of encrypted and VPN traffic using time-related features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP)*, pages 407–414, Rome, Italy, 2016.

[21] L. Grimaudo, M. Mellia, and E. Baralis. Hierarchical learning for fine grained Internet traffic classification. In *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 463–468, Aug 2012.

[22] L. Grimaudo, M. Mellia, E. Baralis, and R. Keralapura. SeLeCT: Self-learning classifier for internet traffic. *IEEE Transactions on Network and Service Management*, 11(2):144–157, June 2014.

[23] Ivo Grondman. Identifying short-term periodicities in internet traffic. *BSc thesis for Applied Mathematics & Telematics University of Twente, Faculty of EEMSC*, 2006.

[24] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, Mar 2001.

[25] J.R.Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[26] M. Laner, P. Svoboda, and M. Rupp. Detecting M2M traffic in mobile cellular networks. In *IWSSIP 2014 Proceedings*, pages 159–162, May 2014.

[27] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[28] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis. In *Proceedings of the Symposium on Applied Computing*, SAC '17, page 506–509, New York, NY, USA, 2017. Association for Computing Machinery.

[29] Sheau-Dong Lang Mian Zhou. Mining frequency content of network traffic for intrusion detection. *Proceedings of the Communication, Network, and Information Security*, 2003.

[30] Phil Wilson Michael E. Raynor. Beyond the dumb pipe: The iot and the new role for network service providers. Technical report, Technical report, Deloitte University.

[31] L.Grimaudo M.Mellia and E.Baralis. Hierarchical learning for fine grained internet traffic classification. In *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*.

[32] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical Report RR-05-13, Queen Mary University of London, Department of Computer Science, 2005.

[33] Alex X. Liu Jeffrey Pang Muhammad Zubair Shafiq, Lusheng Ji and Jia Wang. A first look at cellular machine-to-machine traffic: Large scale measurement and characterization. In *SIGMETRICS Perform. Eval. Rev.*, 2012.

[34] Alessandro Finamore Marco Mellia Michela Meo Maurizio M. Munafò and Dario Rossi. Experiences of internet traffic monitoring with tstat. experiences of internet traffic monitoring with tstat. In *IEEE Network, 25(3)*.

[35] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys Tutorials*, 21(2):1988–2014, November 2019.

[36] Vibha Pant, Roberto Passerone, Michele Welponer, Luca Rizzon, and Roberto Lavagnolo. Efficient neural computation on network processors for IoT protocol classification. In *Proceedings of the First New Generation of Circuits and Systems Conference*, NGCAS 2017, Genova, Italy, September 7–9, 2017.

[37] Michela Meo Dario Rossi Paola Bermolen, Marco Mellia and Silvio Valenti. Kiss: Stochastic packet inspection classifier for udp traffic. In *IEEE/ACM Transactions on Networking*, 2010.

[38] Anilkumar Patro. Weka - modified for data mining course at wpi, 2004. `http://davis.wpi.edu/xmdv/weka/`.

[39] Niclas Persson. Event based sampling with application to spectral estimation. *Division of Control & Communication Department of Electrical Engineering Linkopings universitet, SE–581 83 Linkoping, Sweden*, 2002.

[40] Antônio J. Pinheiro, Jeandro de M. Bezerra, Caio A.P. Burgardt, and Divanilson R. Campelo. Identifying IoT devices and events based on packet length from encrypted traffic. *Computer Communications*, 144:8 – 17, 2019.

[41] Arun Swami Rakesh Agrawal, Christos Faloutsos. Efficient similarity search in sequence databases. In *FODO '93: Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, October 1993.

[42] CAROLINA RUIZ. Cs 525d knowledge discovery and data mining project 2: Sequence mining, 2004. `http://web.cs.wpi.edu/~cs525d/s04/Projects/project2.html`.

[43] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang. Large-scale measurement and characterization of cellular machine-to-machine traffic. *IEEE/ACM Transactions on Networking*, 21(6):1960–1973, Dec 2013.

[44] Muhammad Zubair Shafiq, Lusheng Ji, Alex X. Liu, Jeffrey Pang, and Jia Wang. A first look at cellular machine-to-machine traffic: Large scale measurement and characterization. *SIGMETRICS Perform. Eval. Rev.*, 40(1):65–76, June 2012.

[45] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar. IoT devices recognition through network traffic analysis. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5187–5192, Dec 2018.

[46] Alberto Dainotti Antonio Pescapè Alessandro Finamore Silvio Valenti, Dario Rossi and Marco Mellia. Data traffic monitoring and analysis chapter reviewing traffic classification. In *Springer-Verlag, Berlin, Heidelberg*, pages 123–147, 2013.

[47] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, Aug 2019.

[48] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Characterizing and classifying IoT traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564, May 2017.

[49] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Characterizing and classifying IoT traffic in smart cities and campuses. In *Proceedings of IEEE INFOCOM Workshop SmartCity, Smart Cities Urban Comput.*, Atlanta, GA, USA, May 2017.

[50] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy. DEFT: A distributed IoT fingerprinting technique. *IEEE Internet of Things Journal*, 6(1):940–952, Feb 2019.

[51] USC/LANDER project. Iot devices' first-time bootup traces, predict id: Usc-lander/iot_bootup_traces-20161207. http://www.isi.edu/ant/lander, 2018.

[52] USC/LANDER project. Iot devices' first-time bootup traces, predict id: Usc-lander/iot_bootup_traces-20181107. http://www.isi.edu/ant/lander, 2018.

[53] Chang Liu Zigang Cao Zhen Li Gang Xiong. Lafft: Length-aware fft based fingerprinting for encrypted network traffic classification. In *2018 IEEE Symposium on Computers and Communications (ISCC)*.

[54] C. Xu, S. Chen, J. Su, S. M. Yiu, and L. C. K. Hui. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Communications Surveys Tutorials*, 18(4):2991–3029, Fourthquarter 2016.

[55] Y. Yang, K. Zheng, C. Wu, and Y. Yang. Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. *Sensors*, 19(11):2528, 2019.