**NTNU – Trondheim**
Norwegian University of
Science and Technology

# Detecting DNS tunneling using machine learning

**Terje Kristoffer Skow**

Submission date: December 2015
Responsible professor: Than Van Do, ITEM
Supervisor: Hai Ngyuen, Telenor Research

Norwegian University of Science and Technology
Department of Telematics

## Abstract

# Preface

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Acronyms

**C&C** Command and control.

**DNS** Domain Name System.

**DPI** Deep Packet Inspection.

**MCD** Minimum Covariance Determinant.

**OCSVM** One-Class SVM.

**RR** Resource Record.

**SVM** Support Vector Machine.

**VPN** Virtual Private Network.

# Chapter 1
# Introduction

Domain Name System (DNS) tunneling has been, and still is, an effective way to sneak around a pay wall to use internet at hotels and cafés for free. It is also a good way to send commands undetected in a Command and control (C&C) attack or exfiltrate data from a locked secure network. With the mobile network becoming an all-IP network DNS tunneling is becoming a threat to the mobile network as well. The users is able to use a tunnel to reduce the data traffic and use services not paid for. If there is using your mobile as a hotspot is a service the carrier charges for using a DNS tunnel on your computer connected to the hotspot will not get you charged. There also exists tools for mobile phones to tunnel IP over DNS to get away from paying for data services completely. This is bad for the carriers, so in this project it is looked at how to detect a DNS tunnel using machine learning. Detecting DNS tunnels has been written about earlier [Far13], but there is still no good way of doing it and the mobile network is a new scene for this problem.

## 1.1 Structure

This report will start with a more complete overview of DNS in chapter 2 and then explain how a DNS tunnel set up and used in chapter 3. In chapter 4 will there be an overview of detection techniques previously tested. Next is an overview of machine learning in chapter 5, in chapter 6 will the experiments be explained and the results be presented. Chapter 7 contains the conclusion and thoughts about future work.

# Chapter 2
# DNS

## 2.1 Introduction

DNS is an important part for the internet. It is a system of distributed databases which contains the information about all the domains. In the mid and late 1980s did the previous system, `HOST.TXT`, encounter problems [MD88] which lead to the creation and standardizing called DNS. Since that has the DNS system been updated and configured many times. It needed to be able maintain a fast response time as the database grew larger, this was solved by using a hierarchical set up. This means that each server only has a limited information and sends the request to a new server until it reaches the correct server. It started with one root server, which has expanded to 13 today. The each layer of the hierarchy is called a zone, and it delegates the responsibility for underlying zones delimited by the `dot` in the request name Figure 2.1.

## 2.2 Structure

The data in the databases are called RR and contains the information about what the server do with the request. It has the following fields [Moc83]:

- NAME – the owner name of the record.
- TYPE – what type of record this is, name-to-IP (A) or IP-to-name (PTR).
- CLASS – define the class of the record, usually `IN` for internet. It is not widely use and not important for this paper.
- TTL – an integer which says how long the record should be cached by the server receiving the response.
- RDLENGTH – Specifies the length of the payload in number of octets. One octet is one octet of bits which corresponds to one character

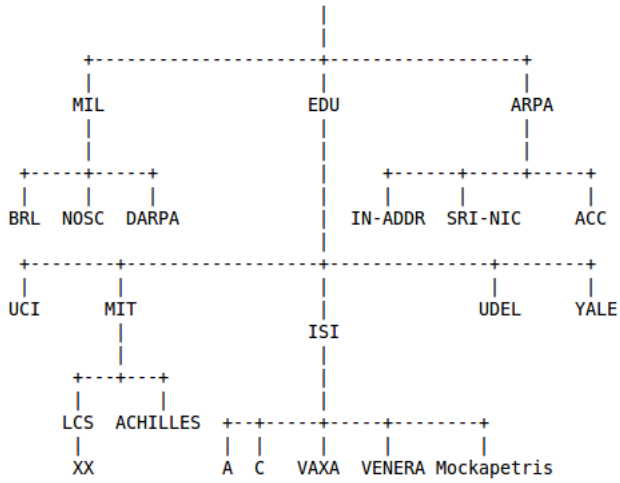



**Figure 2.1:** Example of name spaces of a root with MIL, EDU and ARPA as immediate subdomains. Each leaf is a domain [Moc87].

– RDATA – the payload of the record. The format and length varies depending on the TYPE and CLASS of the RR.

DNS was first implemented with around 15 different RR TYPE, which has now increased to over 30 [Far13] as a result of the development of the internet. The most notable values for TYPE are:

– A – the payload will contain the ipv4 address of the hostname requested. This is the most used TYPE

– AAAA – contains the ipv6 address of the hostname.

– CNAME – canonical name, respond with the correct alias of the hostname.

– MX – the mail exchange for the domain

– TXT – a text response with large payload, can be used in many ways and are an important type in DNS tunneling.

– PTR – pointer record. Used to map a hostname to an IP-address, commonly known as a reverse lookup.

– NS – authoritative name server for the domain

The 'A' type RR for telenor.no at the name server will look something like this:

| Field | Value |
|---|---|
| NAME | telenor.no |
| TYPE | A |
| CLASS | IN |
| TTL | 300 |
| RDLENGTH | 15 |
| RDATA | 153.110.156.145 |

**Table 2.1:** Example of RR for telenor.no

## 2.3 How it works

To explain of DNS works is an example the easiest way

When a request goes through DNS it starts in the root zone, where it sent down the hierarchy to the `.no` zone.

Normally a DNS server in an enterprise does not send requests directly to the internet, but use an internal DNS server instead. If you are the owner of the authoritative server for a domain, you can control the responses. This is what a DNS tunnel exploits, which will be explained more in the next section.

# Chapter 3
# DNS Tunneling

DNS tunneling was first used by people who exploited that DNS was not monitored in network you had to pay to use, e.g. hotels and cafés. It was used as an Virtual Private Network (VPN) tunnel. In later years it has been discovered that in enterprises the DNS are not monitored as much as other traffic on the network. People has therefore figured out that it is a good way to ex filtrate data in secure networks. DNS could also be used for a "command and control" attack, where commands are sent over DNS.

The way DNS works it that if you control the authoritative DNS server for a domain you can easily send commands.

With the increase of smartphones it has been discovered that DNS tunneling could again be used as the it started, to use the network without having to pay for it. Carriers can not start charging for regular queries since just regular use of a the internet produces a lot of DNS traffic. Which an user would not see and it would be hard for the carrier to explain for an user what he has been charged for.

# Chapter 4
# DNS Tunneling Detection

There has been done some research in detecting DNS tunneling over the years, but as it is still a problem no one has found a solution that is cost efficient. The best way for detecting tunnels is still Deep Packet Inspection (DPI) which is slows down the DNS requests as the amount of requests increase. DPI looks into each request and response for payload information which can indicate a DNS tunnel. For instance if requests maximizes the size of the labels and the overall name it should be looked at [Far13], this since tunnels would try to minimize the number of packages and maximize speed. Looking at the hostname should also be an indication since regular DNS names is dictionary words or have some meaning, while an encoded name would be meaningless. Traffic analysis is the other main alternative to detecting tunnels. Looking at volume, frequency and other attributes of DNS traffic could give indication of a tunnel. Earlier research has covered different techniques, looking at the volume of DNS traffic from a IP address or the volume of DNS traffic to a specific domain [Far13]. The overarching way of detecting tunnels with traffic analysis is looking for anomalies and stand out cases.

## 4.1 Traffic analysis

Data that is tunnelled through DNS is normally limited to 512 bytes per request, which leads to clients to send and receive lots of requests and responses. If the server should have the possibility to send data to the client will the client have to constantly send requests to get the data as a response from the server. All this leads to lots of DNS traffic which is not similar to normal use.

# Chapter 5
# Machine Learning

Machine learning is a way of using statistics to solve problems either by learning from a data set how what the output should be for an input or by figure out different patterns in the data. This is called supervised and unsupervised learning respectively. It is widely used in spam filtering and search engines.

## 5.1 The Basics

The basics for machine learning is to use to the computer to create a model from which the computer is able to predict the output or category of an input based on the values of the input. Machine learning most often consists of two phases, training and testing. In the training phase does the classifier model analyse the dataset given to work out how to categorize the observation and find patterns. Testing is where a new dataset is used to see how accurate the algorithm is. It is normal to divide a dataset in two for these phases, using no less than 50 percent in the training phase. The training can be done either supervised or unsupervised. Supervised learning means the data has a label of which it is meant to be categorized as, and unsupervised use unlabeled data with the assumption that the majority of data is considered normal. In this project the data were unlabeled so the training had to be done unsupervised

## 5.2 Anomaly Detection

Anomaly detection, also called outlier detection, is a problem very suited for machine learning. It is a way of identify observations or data which doesn't fit an expected pattern. These observations will be referred to as anomalies or outliers in this report. Illustrated in Figure 5.1 is an example of anomalies in a two-dimensional data set. $N_1$ and $N_2$ is the normal areas where most of the observations are, $o_1$ and $o_2$ are anomalies and $O_3$ is an area with multiple anomalies.

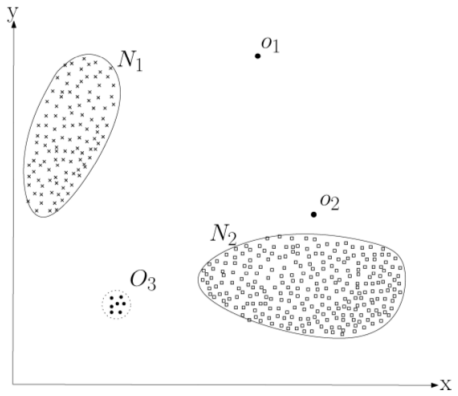



**Figure 5.1:** Example of anomalies in a 2D dataset[CBK09].

## 5.3 Scikit-learn

Scikit-learn was chosen as the machine learning library in this project. It is a Python library and I am most comfortable programming in Python. Alternatively could either Weka or R be used. Weka is a complete program with GUI, and has an API making it possible to integrate in a Java program. R is a special programming language which is specifically designed for statistical analysis. Scikit-learn depends on `numpy` and `scipy` to have the data in correct arrays and to do statistical analysis, and to create graphs is it necessary to use `matplotlib`. These do not follow in the regular installation of scikit-learn, and will have to be install on its own.

### 5.3.1 Models

Support Vector Machines (SVMs) is general terms for models that use supervised learning to analyse and recognize patterns in the data given as a training set. The problem must be binary, which means each point must belong to one of two categories. OCSVM is a unsupervised SVM model, which is used used to solve outlier detection problems. In this project that was needed since the data was not labeled. Elliptical envelope was also used to see the difference between machine learning models. It is a covariance model, and is used to calculate a Minimum Covariance Determinant (MCD). MCD is a function that tries to find a proportion value of the correct observations which is then used to weight the observation to give a better representation [PVG+11].

# Chapter 6
# Results

With scikit-learn was a program created in python that reads through the dataset and puts the DNS calls in to an ndarray, which is a N-dimensional array. This is to easier use to right values for the machine learning model. The program also cleans up the dataset by removing some of the fields, which for this project seemed unnecessary. I created multiple scripts for taking out different sections of the dataset, to see if there would be any difference. The code for the parser program is in A

The main program read through a `csv` file, the dataset, and put each line as an array in a ndarray. N-dimensional array is a numpy class which is beneficial to use with scikit-learn. Further the ndarray was preprocessed to scale everything to a similar level. This level is set by calculating the mean and standard deviation of the ndarray.

The different classifiers are initiated with parameters that is changed to see what best fit the data. As the data is unlabeled, it is impossible to know how many of the observations are a part of a DNS tunnel if there is any. The classifiers must not have any inputs, but it will help making them more precise. In this project the contamination level, which is the level of data which is viewed as incorrect, was first calculated as the percentage of observations which was over average as a base. The value has been change up and down, but kept in the same area.

To be able to show the solutions and really understand what the machine learning model did, all of the experiments used two dimensions. The dimensions change between $downlink, uplink, duration, downlink/duration$ and $uplink/duration$. The values was not marked with any type, but based on network theory is duration set to the of length of conversation in seconds. The uplink and downlink is the number of bytes transferred up to the server or downloaded from the server, respectively.

Since it was no known DNS tunnels in the dataset, it is impossible to use precision, recall or the $F_1$ measure as results. The precision and recall measurements are defined

in algorithm 6.1. The $F_1$ measurement was defined van Rijsbergen [MY02] as a way of combining precision and recall. The formula is in algorithm 6.2. As seen from the definitions without the knowledge of tunnels in the dataset will there not be a way of measuring the correctness of the machine learning algorithms.

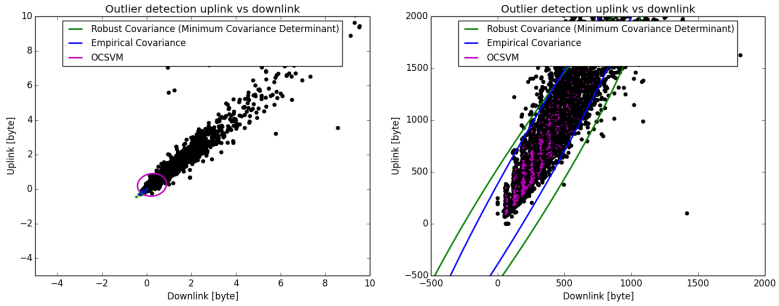
**Algorithm 6.1** Recall and precision definitions

$$recall = \frac{Numberofitemsofacategoryidentified}{Numberofitemsinthecategoryinthedataset}$$

$$precision = \frac{Numberofitemsofacategoryidentified}{Numberofitemsclassifiedtothecategory}$$

**Algorithm 6.2** $F_1$ measure

$$F_1(R, P) = \frac{2RP}{R + P}$$




**(a)** Scaled data **(b)** Unscaled data

**Figure 6.1:** Difference between scaled and unscaled data

## 6.1 Scaled data vs unscaled data

As seen in Figure 6.1 the scaled and unscaled graphs bears the same characteristics. Most of the observations has almost the same value for uplink and downlink. The results were better for OCSVM with scaled data, while the covariance calculations worked better with the unscaled. The unscaled data spreads so far out that it is hard to see all the data points, while the scaled data is more compact giving a better overview. The ellipses on the figures is the learned decision of the classifier model. Meaning that inside the ellipses is the area where an observation is considered normal. Since the OCSVM is the main focus, scaled data was used for the rest of the experiments.

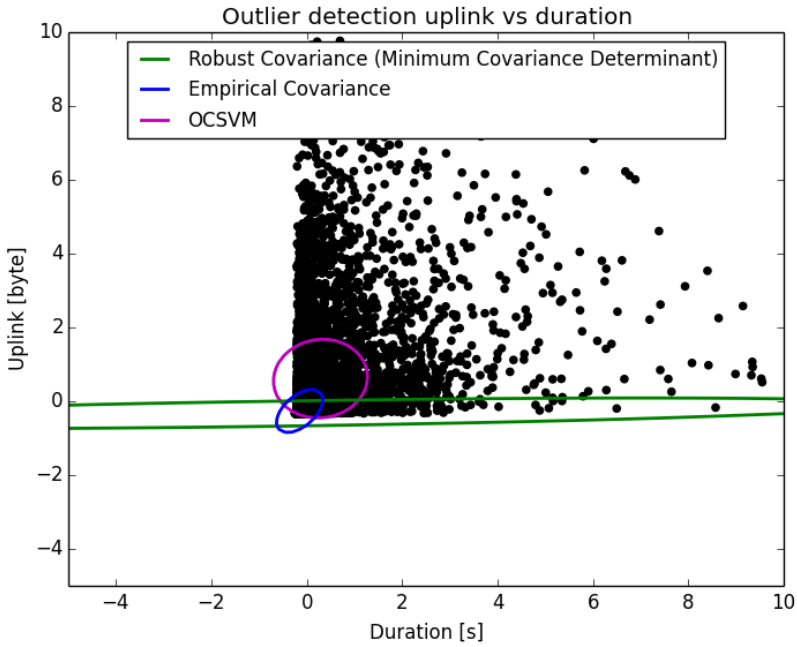



**Figure 6.2:** Uplink vs duration of session.

## 6.2 Different axes

Next up was looking which values would be best for the axes to represent the data. In Figure 6.1 the fields of the dataset used were `uplink` and `downlink`. This was the first thoughts of finding irregularities, as mentioned in chapter 4 is this a volume analysis comparing total volume from a user in one DNS session. The experiment were then ran with different values to see the difference. In Figure 6.2 the size of the uploaded data were compared to the time the session were used, this shows that there are a number of users in the same area which the model learn as the normal area, which in scale is around 0. The result were similar when changing uplink with downlink. There are users uploading and downloading lots of data in short time, this lead to the idea of introducing a new field to the dataset. Combining the discoveries from Figure 6.1a and Figure 6.2, by seeing how much data where uploaded or downloaded during a session. This is depicted in Figure 6.3. In this graph it is possible to see that the observation is even more clustered. This seemed like a good representation.

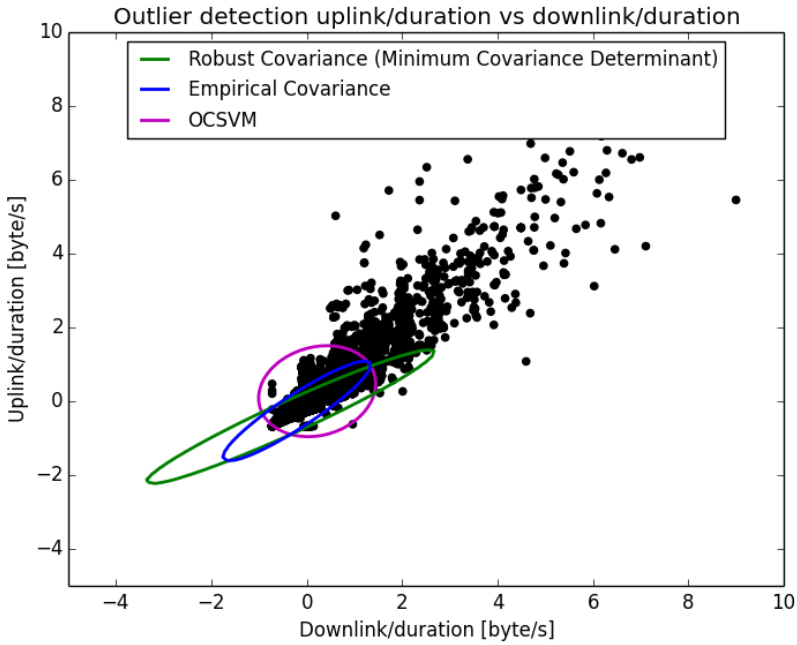



**Figure 6.3:** Average speed in each direction

## 6.3 Different parameters

The fields of the dataset is, as mentioned earlier, not the only variable when finding the best way for a model to fit to the dataset. The machine learning model has parameters that is given when instantiated. For OCSVM is it possible to set what kind of kernel it should use and the number of training errors among others. `Kernel` is the function the model should use. The number of training errors, `nu` in scikit-learn, is the upper bound fraction of outliers in the training set, and the lower bound of training examples used as support vectors. For this project the kernels used where `rbf` and `linear` where rbf is the one used in the graphs so far. The nu was first set to 0.15 which was calculated as the fraction of observations above average, this has been changed to see how it affect the learned decision. In 6.4 do you see how changing nu is affecting the model. It is clear to see that in this dataset there are a low fraction of observations that does not fit the model. Using the linear kernel the model tries to find a linear line it can draw where all observations on one side is correct observations and on the other side is outliers. This is seen in . It was worth a try, but it seems that `rbf` if the best kernel.

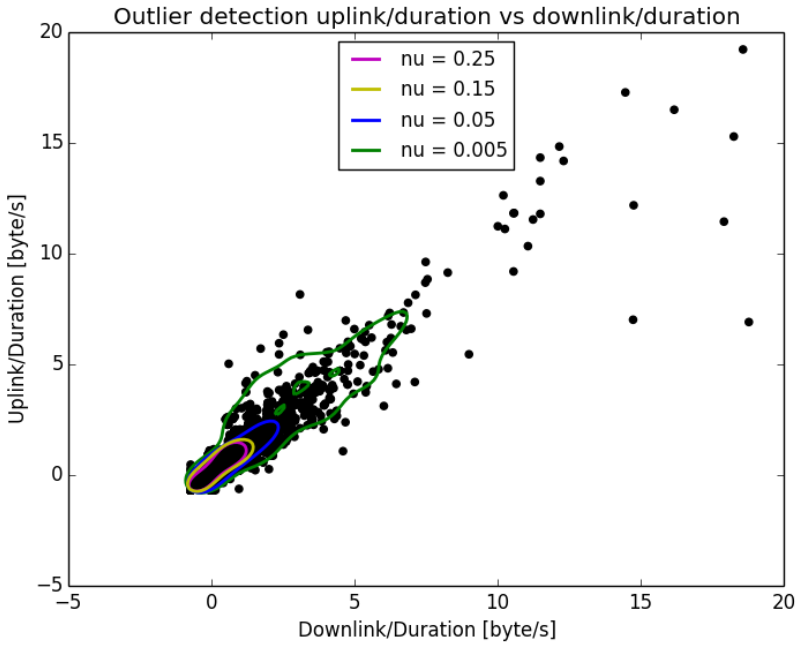



**Figure 6.4:** Changing the nu parameter of the OCSVM

## 6.4 Predicting

The dataset were also ran through a predicting program where 75 percent of the dataset were used to train the classifier and the last 25 percent were used to test it. Which means that the model tried to give around 5000 observations the correct label. Here the classifier had no problems using more then two features, so the classifier were trained with different features to see how it worked. As seen in table 6.1 is it seen that no matter the features are the predictions quite similar. The one thing to notice is that the more feature used is resulting in a little higher number of outliers, but the change from 33 to 38 out of 5167, is only 0.1 percent change. With $nu = 0.16$ is the percent of outliers around 10 percent, while with $nu = 0.005$ is it only 0.6 percent. OCSVM with linear kernel were also tested with prediction but were not nearly as accurate as the rbf kernel.

| | nu = 0.16 | | nu = 0.005 | |
|---|---|---|---|---|
| Features | Outliers | Inliers | Ouliers | Inliers |
| $Uplink, downlink$ | 438 | 4729 | 29 | 5138 |
| $Uplink, duration$ | 557 | 4610 | 28 | 5139 |
| $Downlink, duration$ | 579 | 4588 | 33 | 5134 |
| $Uplink, downlink, duration$ | 568 | 4599 | 38 | 5129 |
| $\frac{Uplink}{duration}, \frac{downlink}{duration}$ | 469 | 4698 | 26 | 5141 |
| $Uplink, downlink, duration, \frac{Uplink}{duration}, \frac{downlink}{duration}$ | 613 | 4554 | 36 | 5131 |

**Table 6.1:** Number of observations classified as outliers and inliers with different features and nu value

# Chapter 7
# Conclusion

## 7.1 Conclusion

With the results the experiments have yielded, seen in chapter 6, does it looks like machine learning is a good way of detecting DNS tunnels. The classifier model OCSVM with the default kernel `rbf`, does create a good decision function which the graphs shows. The table 6.1 shows that the predicting function also gives good results. Even though it is not possible to say if it is completely correct. The dataset is unlabeled and might not contain a DNS tunnel, but the results gives indication of sessions and users which should be inspected. The Elliptical envelope model has the disadvantage that it has to create a ellipse as the decision function which seems to make inaccurate either making the decision function cover a very large or too small area. As seen in 6.3 mot the Robust Covariance and the Empirical Covariance cover smaller area of observations and larger area of blank space than the OCSVM. The linear kernel of OCSVM finds a linear function in which observation on one side is the inliers and the other side is the outliers. This showed some results with some of the features, but was worse than the rbf kernel overall.

## 7.2 Future work

With the findings in this project could it be interesting doing experiments where with data know to contain a DNS tunnel, either continue with unlabeled learning or label the data and to supervised learning. This could also be tested against one another. Also creating a program which could take in a real time data stream and give out suspicious connection based on this model or the best model from the previous experiment mentioned, making it a complete program for tunnel detection. There are also more to data to look at to make a complete detection program. The dataset contained also had information about other services the user did. Combining more features might be of help, e.g. look if an user makes a `http` request after the DNS session or what kind of data plan the user has.

# References


[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[Far13] Greg Farnham. Detecting dns tunneling. *InfoSec Reading Room*, 2013.

[MD88] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 123–133, New York, NY, USA, 1988. ACM.

[Moc83] Paul V Mockapetris. Domain names: Implementation specification. 1983.

[Moc87] Paul V Mockapetris. Domain names-concepts and facilities. 1987.

[MY02] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *the Journal of machine Learning research*, 2:139–154, 2002.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

# Appendix A
# Parser program

```
import csv

def allDnsCalls(csvIn):
        #opens the outputfile
        with open('allDnsCalls.csv', 'wb') as writefile:
                spamwriter = csv.writer(writefile, delimiter=',', quotechar='|')
                for row in csvIn:
                        #Remove values though as unecessary for the experiment
                        row.pop(-1)
                        row.pop(-2)
                        row.pop(-2)
                        #9000 where the representing value for a DNS call
                        if row[0] == '9000':
                                row.pop(0)
                                #calculate uplink/duration and downlink/duration
                                if float(row[-1]) == 0:
                                        bpsUp = row[1]
                                        bpsDown = row[2]
                                else:
                                        bpsUp+ = float(row[1])/float(row[-1])
                                        bpsDown = float(row[2])/float(row[-1])

                                row.append(bpsUp)
                                row.append(bpsDown)
                                spamwriter.writerow(row)

#Opens the dataset file
with open('ggsnSample-4Kristofer-hashIMSI.csv','rb') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
```

```
        allDnsCalls(spamreader)
```

# Appendix B
# Plot program

```
import numpy as np
import csv
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
from sklearn import preprocessing

with open('imsiOnlyOnce.csv','rb') as csvfile:
  spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
  temp = []
  for row in spamreader:
    one = float(row[4])
    two = float(row[5])

    temp.append([one,two])

  X2 = np.array(temp)
  X1 = preprocessing.scale(X2)


classifiers = {
    #"linear nu = 0.005": OneClassSVM(kernel='linear', nu=0.005),
    "nu = 0.15": OneClassSVM(nu=0.15),
    "nu = 0.25": OneClassSVM(kernel='rbf', nu=0.25),
    "nu = 0.05": OneClassSVM(nu=0.05),
    "nu = 0.005": OneClassSVM(nu=0.005)}
colors = ['m', 'g', 'b', 'y']
legend1 = {}
```

```
# Learn a frontier for outlier detection with several classifiers
xx1, yy1 = np.meshgrid(np.linspace(-5, 20, 200), np.linspace(-5, 20, 200))
with open('Data/write.txt','wb') as writefile:
  for i, (clf_name, clf) in enumerate(classifiers.items()):
      plt.figure(1)
      clf.fit(X1)
      Z1 = clf.decision_function(np.c_[xx1.ravel(), yy1.ravel()])
      Z1 = Z1.reshape(xx1.shape)
      legend1[clf_name] = plt.contour(
          xx1, yy1, Z1, levels=[0], linewidths=2, colors=colors[i])

legend1_values_list = list( legend1.values() )
legend1_keys_list = list( legend1.keys() )

# Plot the results (= shape of the data points cloud)
plt.figure(1)  # two clusters
plt.title("Outlier detection uplink/duration vs downlink/duration")
plt.scatter(X1[:, 0], X1[:, 1], color='black')
plt.xlim((xx1.min(), xx1.max()))
plt.ylim((yy1.min(), yy1.max()))

plt.legend((legend1_values_list[0].collections[0],
            legend1_values_list[1].collections[0],
            legend1_values_list[2].collections[0],
            legend1_values_list[3].collections[0]),
           (legend1_keys_list[0], legend1_keys_list[1],
           legend1_keys_list[2], legend1_keys_list[3]),
           loc="upper center",
           prop=matplotlib.font_manager.FontProperties(size=12))
plt.ylabel("Uplink/Duration [byte/s]")
plt.xlabel("Downlink/Duration [byte/s]")

plt.show()
```

# Appendix C
# Prediction program

```
import numpy as np
import csv
from sklearn import preprocessing
from sklearn import svm

with open('trainset.csv','rb') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
        temp = []
        for row in spamreader:
                #append the wanted values to an ndarray and scale it
                one = float(row[1])
                two = float(row[2])
                three = float(row[3])
                temp.append([one,two,three])

        data = np.array(temp)
        data = preprocessing.scale(data)

        #initiate and train the model
        oneSVM = svm.OneClassSVM(nu=0.005)
        oneSVM.fit(data)


with open('testset.csv','rb') as csvfile2:
        spamreader = csv.reader(csvfile2, delimiter=',', quotechar='|')
        temp = []
        #get the wanted values from the testset into an ndarray and scale it
        for row in spamreader:
                row[4] = float(row[1])
```

```
                row[5] = float(row[2])
                three = float(row[3])
                temp.append([one,two,three])

        data = np.array(temp)
        data = preprocessing.scale(data)

        #run the test, returns a ndarray with 1 for inlier and -1 for outlier
        prediction = oneSVM.predict(data)

        outlier = 0
        inlier = 0
        for x,i in enumerate(prediction):
                if i < 0:
                        negative += 1
                else:
                        positive += 1
        print positive, negative
```