Universitat Oberta de Catalunya

uoc.edu

Detector de Covert Timing Channels basat en Machine Learning

Patrik Martínez Villamarín
Màster Universitari de Seguretat de les Tecnologies de la Informació i de les Comunicacions
Anàlisi de dades

Director del TFM: Daniel Lerch Hostalot
Professora responsable de l'assignatura: Helena Rifà Pous

Data d'entrega: 2 de Juny de 2020


Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

AGRAÏMENTS

A na Maria Rosa, per
respectar-me, ajudar-me i estimar-me.

A na Gal·la i en Joel, per obrir-me els ulls.

Als meus pares i germana, per
estar sempre presents.

Al Julian, per creure en jo.

Llicències alternatives (triar alguna de les següents i substituir la de la pàgina anterior)

A) Creative Commons:

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-CompartirIgual 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-SenseObraDerivada 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-CompartirIgual 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement 3.0 Espanya de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © 2020 Patrik Martínez.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3

or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

© Patrik Martínez
Reservats tots els drets. Està prohibit la reproducció total o parcial d'aquesta obra per qualsevol mitjà o procediment, compresos la impressió, la reprografia, el microfilm, el tractament informàtic o qualsevol altre sistema, així com la distribució d'exemplars mitjançant lloguer i préstec, sense l'autorització escrita de l'autor o dels límits que autoritzi la Llei de Propietat Intel•lectual.

FITXA DEL TREBALL FINAL

Títol del treball:	*Detector de Covert Timing Channels basat en Machine Learning*
Nom de l'autor:	*Patrik Martínez Villamarín*
Nom del consultor/a:	*Daniel Lerch Hostalot*
Nom del PRA:	*Helena Rifà Pous*
Data de lliurament (mm/aaaa):	*06/2020*
Titulació o programa:	*Màster Universitari de Seguretat de les Tecnologies de la Informació i de les Comunicacions*
Àrea del Treball Final:	*Anàlisi de dades*
Idioma del treball:	*Català*
Paraules clau	*Steganography, Covert Channels, Covert timing channels, Machine learning, Countermeasures*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

L'objectiu principal d'aquest Treball Final de Màster és dissenyar i implementar una eina basada en *machine learning* per a detectar tràfic de xarxa encobert segons el mecanisme d'alteració de l'ordre de PDU.

Amb aquesta motivació, s'ha dissenyat i implementat un mètode no descrit a la literatura que permet ocultar un missatge utilitzant les PDU del protocol RTP per transmetre vídeo i àudio. La funció de distribució del tràfic resultant compleix els criteris per considerar el mètode indetectable segons alguns autors.

A partir del conjunt de dades obtingut del tràfic de xarxa tant amb flux no alterat com amb un que porta un missatge ocult, s'extrauen les característiques amb una adaptació del mètode PPD per entrenar el classificador.

S'utilitzen les característiques obtingudes per entrenar un classificador que es prova en diferents escenaris amb una capacitat de detecció de missatges curts d'un 95,7%.

Abstract (in English, 250 words or less):

The main objective of this Master's Thesis is to design and implement a machine learning-based tool to detect covert network traffic according to the mechanism of alteration of the PDU order.

A method not described in the literature that allows a message to be hidden using the RTP protocol PDUs to transmit video and audio has been designed and implemented. The resulting traffic distribution function meets the criteria to consider the method undetectable.

From the data set obtained from the network traffic with both unaltered flow and one with a hidden message, the characteristics are extracted with an adaptation of the PPD method to train the classifier.

The obtained features are used to train a classifier that is tested in different scenarios with 95,7% detection capacity for short messages.

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

La literatura ha deixat constància que els secrets han format part de forma molt important de la història de la humanitat des dels seus començaments. A la par, els mètodes per mantenir la informació oculta també han anat evolucionant al llarg de la història.

El primer que va descriure mètodes per transmetre missatges ocults va ser l'historiador i geògraf grec *Herodotus dee Halicarnassus* (484 aC – 425 aC) al seu llibre *Las històrias* [1]. El considerat en l'actualitat pare de l'estudi de l'història descriu com van eliminar la cera que servia de fulla a les antigues taules d'escriptura de forma que va quedar la fusta a l'aire per escriure-hi el missatge i el van cobrir novament amb cera de forma que el va ocultar. També descriu com es va rasurar el cap d'un esclau per escriure el missatge a la pell del cap i, quan els cabells van tornar a créixer, l'esclau va partir cap al seu destí.

Una de les formes més utilitzades ha estat dins un escrit de qualsevol natura, com cartes o articles, seleccionar determinades posicions del text conegudes per l'emissor i el receptor per ocultar un missatge dins el text. Aquesta tècnica es coneix pel nom *ranura de Cardano*, en honor al matemàtic italià *Gerolamo Cardano* que la va idear cap a l'any 1550 [2]. No obstant això, no va ser el primer a utilitzar-la. Així, al llibre *Somni de Polífilo* (*Hypnerotomachia Poliphili*, ed. *Aldus Manutius*), escrit per *Francesco Colonna* al 1499, es pot obtenir la frase *Poliam frater Franciscus Columna peramavit* (El germà Francesco Colonna va estimar apassionadament a Polia) si s'extrau la primera lletra de tots els capítols, els quals són 38.

Una altra tècnica molt utilitzada consisteix a fer una marca d'aigua al paper quan aquest està encara humit de forma que després es pugui veure a contrallum quan està eixut [1]. Serveix per verificar el paper moneda, o bitllets, des de l'origen de la seva creació de forma que dificulti la falsificació.

Aquests conjunts de tècniques i moltes altres es van agrupar al voltant del nom esteganografia (*steganos* -ocult- i *graphos* -escriptura) que defineix precisament els mètodes per ocultar la informació dins missatges aparentment inòculs. Dit d'una altra forma, l'esteganografia permet la transmissió d'un missatge secret mitjançant un medi no segur o protegit. Aquesta definició no exclou la criptografia, però tampoc la inclou. És a dir, són dues tècniques diferents però que poden ser complementaries de forma que un missatge es pot encriptar abans de ser compartit utilitzant una tècnica d'esteganografia. Aquesta relació s'exposa al llibre *Steganographia. Hoc est: Ars per ocultam scripturam animi sui voluntatem absentibus aperiendi certa* (Esteganografia o l'art precís de descobrir la voluntat de la seva ànima als quals estan absents per mitjà de l'escriptura oculta) al qual Johanes Trithemius l'any 1500 (encara que publicat més de cent anys després) va descriure nombroses tècniques d'ocultació mitjançant l'esteganografia i la criptografia [2]

En l'actualitat, les possibilitats o tècniques a l'abast de qualsevol al món digital que engloba aquest terme són molt elevades. Així, es poden emprar diferents tipus d'elements contenidors, com aleshores fotografies digitals, fitxers d'àudio i vídeo o protocols de comunicació. Sense cap dubte els més utilitzats són els continguts multimèdia, especialment les imatges, on la capacitat de còmput actual permet implementar tècniques molt avançades les quals, conjuntament amb el volum d'informació que es transmet per Internet, fa que resulti impossible la implementació de mecanismes de detecció a gran escala. Menció especial requereixen les tècniques de *watermarking* que serveixen per ocultar la identitat del propietari o creador d'un document gràfic i que s'utilitzen per oferir seguretat o protegir la propietat intel·lectual.

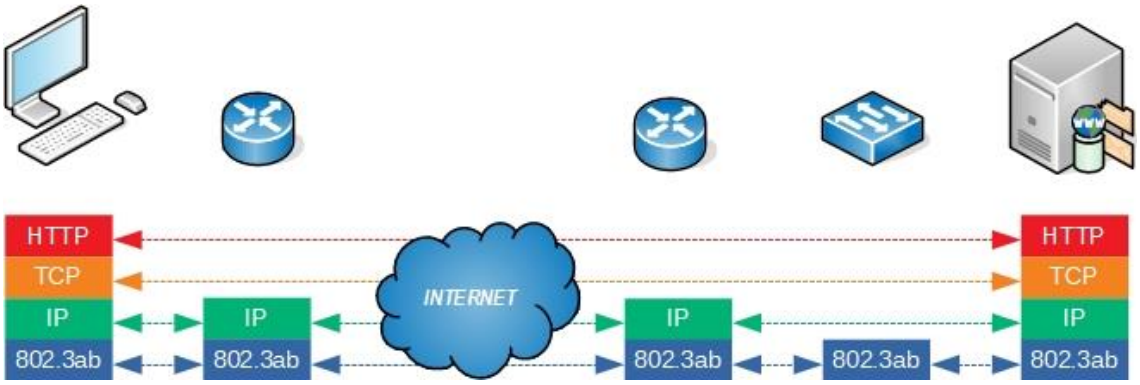
No obstant això, cal tenir present que les tècniques antigues es segueixen utilitzant, tal com es demostra a una carta que va enviar a l'assemblea de San Francisco, als EUA, el que va ser governador de Califòrnia l'any 2009, Arnold Schwarzenegger, per rebutjar un projecte de llei [1].

> To the Members of the California State Assembly:
>
> I am returning Assembly Bill 1176 without my signature:
>
> For some time now I have lamented the fact that major issues are overlooked while many unnecessary bills come to me for consideration. Water reform, prison reform, and heath care are major issues my Administration has brought to the table, but the Legislature just kicks the can down the alley.
>
> Yet another legislative year has come and gone without the major reforms Californians overwhelmingly deserve. In the light of this, and after careful consideration, I believe it is unnecessary to sign this measure at this time.
>
> Sincerely,
> Arnold Schwarzenegger

Si s'agafen les primeres lletres de cada línia, es pot veure un missatge ocult (*Fuck You*) que li va servir al governador com resposta a un afront política que va ocorregut dies abans amb els opositors.

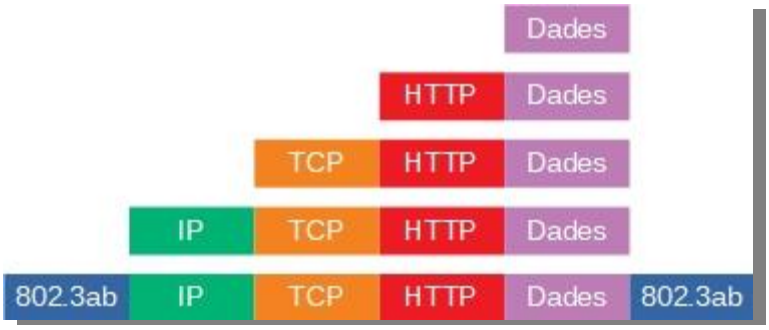
En un altre sentit, és important aclarir el funcionament genèric de les xarxes de telecomunicació per tal que es pugui entendre l'objectiu del treball. Les xarxes de telecomunicació gestionen la informació mitjançant capes que estructuren les responsabilitats a l'hora de transmetre els missatges. D'aquesta forma, la xarxa més coneguda i objecte d'aquest estudi és la xarxa Internet que permet transportar nombrosos tipus de missatges o d'informació gràcies al model anomenat TCP/IP. Sense necessitat d'aprofundir, és important saber que aquest model divideix en quatre capes les responsabilitats, les quals són sempre les mateixes per a cada capa però implementades de formes diferents en forma de distints protocols.

Els protocols per tant són un conjunt de regles que determinen la semàntica i la sintaxi que permet resoldre de forma positiva per a la comunicació les responsabilitats de la capa corresponent. Una necessitat intrínseca a la definició de protocol és que als dos extrems de la mateixa capa que es volen comunicar el protocol ha de coincidir.



Imatge 1.01. Capes amb diferents extrems.

La implementació de cada protocol queda relegada a una sèrie de procediments que es basen en uns missatges estàndards els quals contenen les dades necessàries per fer possible la comunicació. Aquests missatges es diuen de forma genèrica PDU (*Protocol Data Units*) i s'encadenen entre capes afegint la seva informació a la capçalera o a la cua.



Imatge 1.02. Cadena de capçaleres a les PDU del model TCP/IP.

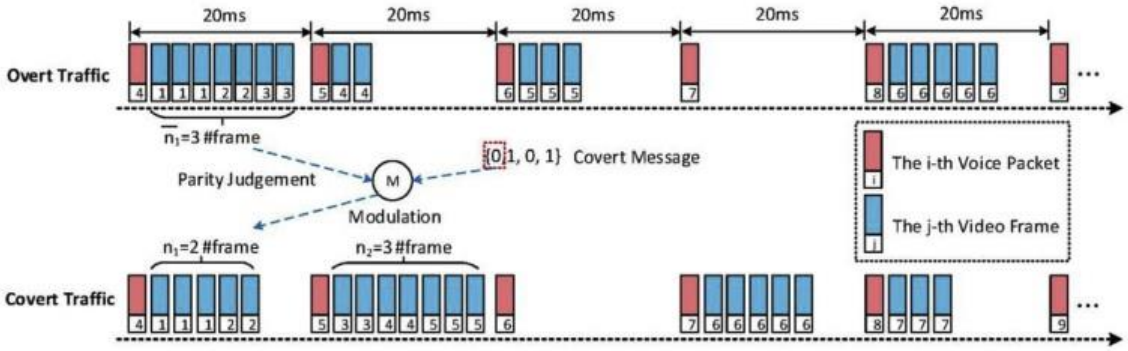
Per tant, l'esteganografia aplicada a les xarxes utilitza els protocols o les seves PDU per ocultar informació creant canals de comunicació anomenats *network covert channels* [4]. L'any 2015, Wendzel et. al. [5] van proposar una classificació de tots els tipus que s'havien detectat i publicat. La comunitat científica va aprovar aquesta classificació que divideix els *network covert channels* en:

- *Covert storage channels*: s'aprofiten les PDU per transmetre la informació oculta, com aleshores camps concrets o format de les dades transmeses.

- *Covert timing channels*: aquells que utilitzen característiques dels protocols, com el temps entre PDU o retransmissions d'algunes en concret.

Aquestes dues categories principals es poden al seu torn classificar segons diferents criteris. Aquest estudi es centrarà en la subcategoria del *covert timing channels* anomenada *Protocol-aware*, és a dir, alteracions basades en el propi protocol, la qual inclou el tipus *Message Ordering (PDU Order)*. Aquest *network covert channel* encobreix la informació utilitzant modificacions de l'ordre del flux de les PDU entre dos o més tipus que requereix el protocol per comunicar-se.

En concret, per aquest treball la generació de les dades es basa a l'aplicació pràctica dissenyada per Zhang et al. [6] a la qual l'ordre dels paquets de vídeo i àudio per a una comunicació VoLTE (*Voice over LTE*) es modifica per paritat tal com es pot veure a la imatge 1.03.



Imatge 1.03. Alteració de l'ordre de les PDU.

L'estegoanàlisi en xarxa, per la seva part, s'encarrega de detectar i/o eliminar aquestes comunicacions encobertes. Debut a la complexitat dels protocols i de les seves PDU, no existeix un estàndard de detecció la qual cosa provoca que hi ha certs tipus d'esteganografia en xarxa o *network covert channels* que poden permetre comunicacions de forma completament opaca per a un possible damnificat. Entre ells està el tipus de *covert timing channel* objecte d'aquest treball. Només hi ha un estudi concret publicat per Wendzel [7] i encara que té una fiabilitat que pot ser

interessant fins i tot pel món real, el mateix autor proposa que s'ha de seguir investigant.

En resum, aquest projecte se centra en aconseguir una evolució de l'estegoanàlisi en xarxa per a la categoria de *covert timing channels* basats a reordenació de PDU.

1.2 Objectius del Treball

L'objectiu principal és detectar *covert timing channels* basats a l'alteració de l'ordre de les PDU quan s'utilitzen dos de diferents utilitzant un classificador de *machine learning*. Seguint la metodologia exposada al punt 1.3, serà molt important per aconseguir-ho dissenyar un mètode adient per aconseguir diferenciar el tràfic que porta missatges encoberts dels quals no en porten.

A la mateixa vegada, caldrà dissenyar i desenvolupar les eines que permetin assolir els següents objectius parcials:

- Generar el tràfic de xarxa que porti missatges encoberts.
- Crear els classificadors, entrenar-los i analitzar els resultats per trobar el millor.
- Validar el seu funcionament en un entorn de tràfic real.
- Obtenir unes conclusions sobre l'efectivitat del classificador.

1.3 Enfocament i mètode seguit

Als *covert timing channels* de tipus reordenació de PDU, la variable a tenir en compte és la relació entre el nombre de PDU dels dos tipus diferents. Per tant, el projecte es basa a la mesura d'aquesta variable i les conclusions en termes de precisió i exactitud d'un algorisme *machine learning* dissenyat per a detectar aquestes anomalies al tràfic de xarxa.

En aquest projecte es treballarà de forma exclusiva amb algorismes de *machine learning* de classificació supervisats, és a dir, algorismes que classifiquen sobre la base de un coneixement après d'un conjunt de dades catalogat. Dit d'una altra forma, cal generar un conjunt de dades prou gran tant de tràfic sense cap informació oculta com tràfic amb un missatge secret i marcar-lo com tal perquè l'algorisme de *machine learning* pugui aprendre a classificar quin tràfic porta un *network covert channel* i quin no. Hi ha molts tipus i un dels objectius és trobar quin és el més adient per l'objectiu principal.

Com ja s'ha explicat, aquest treball es basarà en l'estudi de la reordenació de PDU i es generarà el conjunt de PDU necessari seguint la descripció de l'algoritme per VoLTE publicat per Zhang et al. [6]. El programari, per tant, haurà de poder generar el conjunt necessari dels dos tipus esmentats de forma que es puguin comptar el nombre de les PDU que simulen el tipus àudio entre les quals simulen el tipus vídeo i generar així uns conjunts de dades que siguin adients per analitzar-les amb els algorismes *machine learning*.

Arribat aquest punt, és interessant entendre el principi de indetectabilitat proposat per Cachin [8]. Al seu treball, estableix que una comunicació encoberta és indetectable si l'entropia relativa entre les funcions de probabilitat del tràfic sense missatge ocult i el que sí que en té és un valor molt petit al voltant de zero.

$$D(P_C \parallel P_s) = \sum P_C(i) \log \frac{P_C(i)}{P_S(i)}$$

La diferència estadística s'utilitza en Zhang et al. [6] per demostrar la fiabilitat dels seus algorismes per ocultar informació.

En resum, es treballarà amb hipòtesis basades en el conjunt de dades caracteritzat per complir el principi de indetectabilitat i un algorisme de *machine learning* de classificació supervisat. D'aquí sorgirà un resultat que caldrà avaluar amb un nou conjunt de dades i analitzar si cal modificar els paràmetres de l'algorisme o canviar el tipus perquè els resultats no són millorables. El resultat final serà el classificador més adient per al tipus d'esteganografia en xarxa descrit.

1.4 Planificació del Treball

Tasca 1: Estat de l'art
Fita: 11 de març de 2020

1.1 Estudi dels articles publicats (7 dies).
Aquesta tasca servirà per analitzar els mètodes existents d'estegoanàlisi en xarxa per a *covert timing channels* basats a reordenació de PDU així com la mètrica utilitzada i les raons científiques suporten aquestes decisions.

Tasca 2: Preparació de l'entorn.
Fita: 31 de març de 2020

2.1 Programari gestió de PDU (10 dies)
Disseny i implementació d'una eina que permeti gestionar el tràfic de xarxa tant en temps real com a partir d'una captura en format PCAP. Ha de ficar cadascuna de les PDU a una base de dades per tractar-les posteriorment.

2.2 Programari reordenació de PDU (4 dies)

Implementació del programa basat en la reordenació *VoLTE* de l'article esmentat prèviament. Ha de generar tràfic amb informació oculta com tràfic sense informació oculta.

2.3 Programari classificador (7 dies)
Disseny i implementació de diferents algorismes classificadors per agilitzar la tasca pròpiament d'anàlisi a la tasca posterior.

Tasca 3: Estegoanàlisi.
Fita: 28 d'abril de 2020

3.1 Generació d'hipòtesi i tests (3 setmanes)
Aquesta tasca serveix per dissenyar i provar les hipòtesis creades a partir de l'anàlisi del resultats de les proves prèvies realitzades. És una subtasca que va a l'una de la següent perquè una alimenta a l'altre.

3.2 Anàlisi de resultats (3 setmanes)
Amb l'anàlisi dels resultats, es poden prendre decisions sobre la generació d'hipòtesi i el disseny de proves. És una subtasca que va a l'una de l'anterior perquè una alimenta a l'altre. D'aquesta forma, una anàlisi de resultats donarà un resultat satisfactori i aquest classificador passarà a la següent etapa.

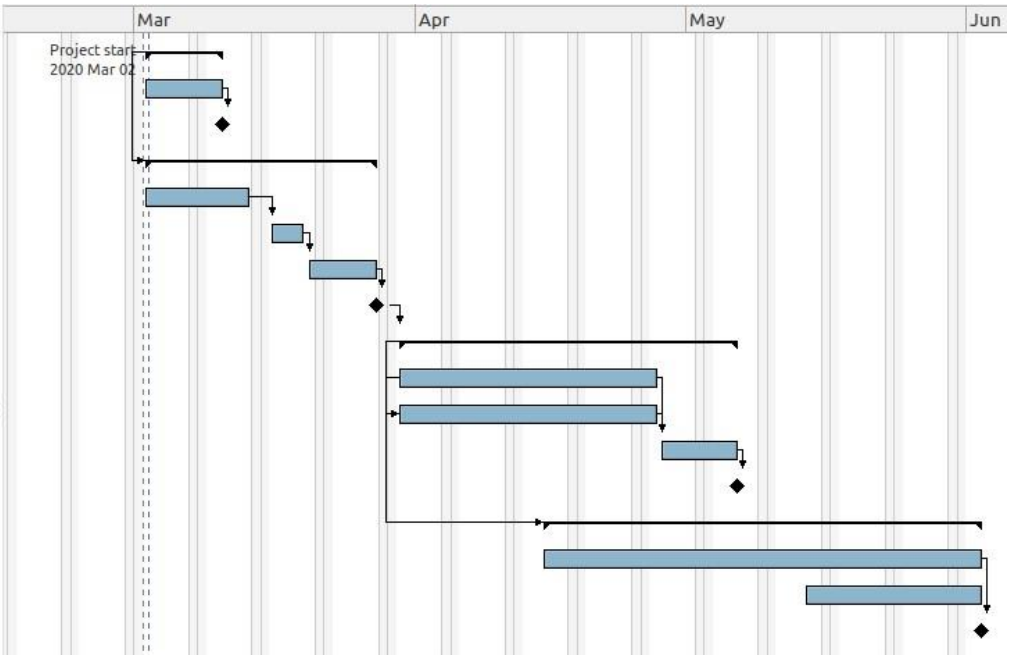
3.3 Prova en entorn real (1 setmana)
Es generarà tràfic dins una xarxa amb nombrosos usuaris per veure si el classificador identifica fluxos de dades amb missatges ocults segons la tècnica de reordenació de PDU amb èxit.

Tasca 4: Redacció
Fita: 2 de juny de 2020

4.1 Redacció de la memòria (3 meses)
Paral·lelament a totes les tasques s'anirà recollint la informació en la memòria d'entrega final.

4.2 Redacció d'un article amb el resultat de la recerca si s'escau (2 setmanes).
Si s'obtingués un resultat que sigui significatiu per a la comunitat, es podria redactar un article per contribuir a l'avanç científic en aquesta matèria.

WBS	Name	Work
1	**Estat de l'art**	**7d**
1.1	Estudi dels articles publicats	7d
1.2	Resum tècniques i mètriques	N/A
2	**Preparació de l'entorn**	**20d**
2.1	Programari gestió de PDU	10d
2.2	Programari reordenació de PDU	4d
2.3	Programari classificador	6d
2.4	Entorn preparat	N/A
3	**Estegoanàlisi**	**49d**
3.1	Generació hipòtesi i tests	21d
3.2	Anàlisi de resultats	21d
3.3	Prova en entorn real	7d
3.4	Classificador preparat	N/A
4	**Presentació de resultats**	**49d**
4.1	Redacció de la memòria	35d
4.2	Redacció d'article (opc)	14d
4.3	Entrega	N/A



Imatge 1.04. Diagrama de Gantt.

1.5 Breu sumari de productes obtinguts

- Memòria: aquest document.
- Eina per a gestionar PDU de xarxa: escrita en *Python* podrà llegir en temps real i d'un fitxer PCAP PDU de qualsevol protocol, catalogar-ho, introduir-lo a una base de dades, crear fluxos i analitzar-los amb algorismes de *machine learning*.
- Eina de reordenació de PDU: escrita en *Python*, simularà el tràfic de xarxa que permet ocultar un missatge en una comunicació amb dues PDU diferents per a un cas concret.
- Base de dades de tràfic de xarxa: es podrà utilitzar per futures investigacions.
- Article d'investigació: si s'escau, es podrà publicar a una revista o fòrum especialitzat.

1.6 Breu descripció dels altres capítols de la memòria

Capítol 2: Esteganografia i estegoanàlisi en xarxa.

Descripció de l'estat actual de l'esteganografia en xarxa i de les contramesures. Servirà per descriure amb profunditat els diferents tipus de *network covert channels*, les seves utilitzacions i l'estat actual de les principals rames d'investigació respecte de la forma d'aturar aquest tipus de tràfic.

Capítol 3: Mètodes de detecció.

Es mostraran les principals característiques dels algorismes classificadors supervisats i les seves aplicacions. Aquest capítol servirà per entendre l'aplicació dels algorismes al problema plantejat i els resultats obtinguts.

Capítol 4: Eines desenvolupades.

Disseny i descripció de les eines desenvolupades per a la realització del projecte. Es mostraran les possibilitats de les eines, el seu propòsit dins el projecte i com han ajudat a obtenir els resultats descrits als objectius.

Capítol 5: Estegoanàlisi.

Descripció dels resultats obtinguts en termes dels objectius definits en el punt 1.2. Aquest capítol presentarà el flux seguit per arribar a les conclusions punt per punt amb les dades objectives que suportin els objectius del projecte.

2. Esteganografia i estegoanàlisi en xarxa

Aquest capítol presenta les diferents tècniques d'ocultació de la informació i les formes que existeixen per mitigar els seus efectes en l'actualitat. S'ha considerat oportú descriure breument també altres tècniques d'ocultació de la informació.

Al món digital actual el primer pensament al parlar d'esteganografia és el món de les imatges i, sense la mateixa rellevància, de fitxers d'àudio i vídeo. L'esteganografia en xarxa és un camp de les tècniques d'ocultar informació encara desconegudes per a la indústria però que evoluciona molt ràpidament al món acadèmic.

Hi ha dos factors que fonamentals per entendre l'atractiu que té aquesta tècnica. Per una banda, la quantitat d'informació que es pot transmetre no està limitada, com aleshores dins un fitxer, ja que es poden transmetre més PDU per fer arribar el missatge ocult. Per l'altra banda, la nul·la atenció de la indústria presenta un escenari perfecte per filtrar informació en entorns laborals o militars, ja que no hi ha cap producte al mercat que pugui controlar si als fluxos de xarxa s'estan transmetent dades ocultes utilitzant l'esteganografia.

No obstant això, establir una comunicació confiable mitjançant esteganografia en xarxa és una tasca complicada deguda a les següents raons [11]:

- Amplada de banda baixa que sol presentar problemes per recuperar errors de transmissió.
- És molt difícil establir canals per retransmissions si es detecten problemes.
- No hi ha sincronització possible.

L'essència de l'esteganografia en xarxa és l'explotació d'alguna o vàries característiques d'un o més protocols utilitzats a qualsevol xarxa de comunicacions. No obstant això, cal que es presentin les següents condicions [12] per parlar amb propietat:

- *C1*: algunes propietats dels protocols de comunicació són modificades.

- *C2*: les modificacions de propietats dels protocols són de dos tipus:

    - *C2a*: relacions amb les imperfeccions dels canals de comunicació, com errors o retards.

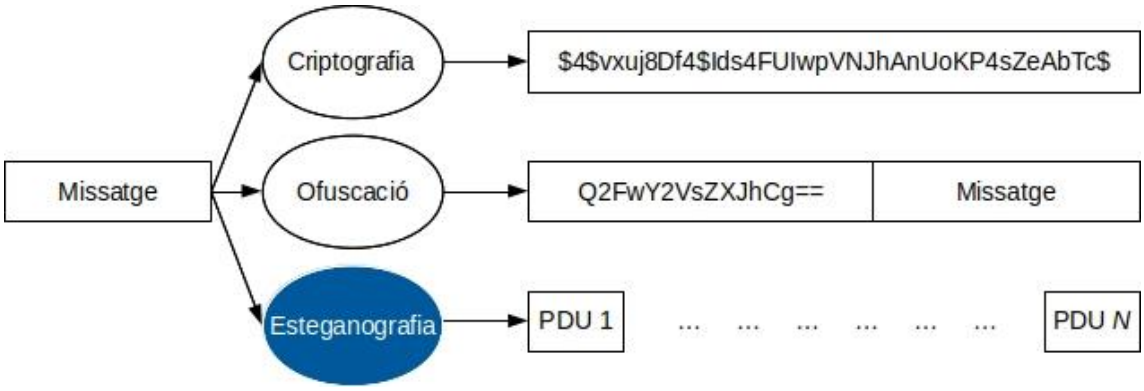
    - *C2b*: definicions del tipus d'intercanvi d'informació, com respostes a peticions o transferències de fitxers, i/o adaptacions de les formes dels missatges, és a dir, fragmentacions o segmentacions.

- *C3*: intent d'emmascarar les modificacions de forma que no siguin fàcils de descobrir, com pot ser intentar que semblin degudes a imperfeccions dels canals de comunicació.

Pel que respecta a aquest projecte, el tipus de modificacions és el *C2b*, ja que el *network covert channel* que es vol estudiar adapta les PDU intercanviades en funció d'unes necessitats específiques. Cal dir que els autors en [12] associen les modificacions de tipus *C2b* com les que es fan als protocols de capes superiors dels models de comunicació, és a dir, els protocols que estan més a prop de les dades de l'usuari i més enfora del nivell físic.

2.1 Altres tècniques de seguretat en les comunicacions

Hi ha dues tècniques que es podrien confondre o interpretar com esteganografia en xarxa o viceversa o la separació de les quals pot semblar difusa. Per una banda hi ha la criptografia i per l'altre les tècniques d'ofuscació del tràfic de xarxa. Les tres, considerant també l'esteganografia en xarxa, tenen com a objectiu protegir informació entre dos comunicadors de forma que cap altra entitat pugui desvetllar-la però hi ha diferències fonamentals que permeten categoritzar cada tècnica de forma adient.



Imatge 2.01. Tipus de tècniques d'ocultació.

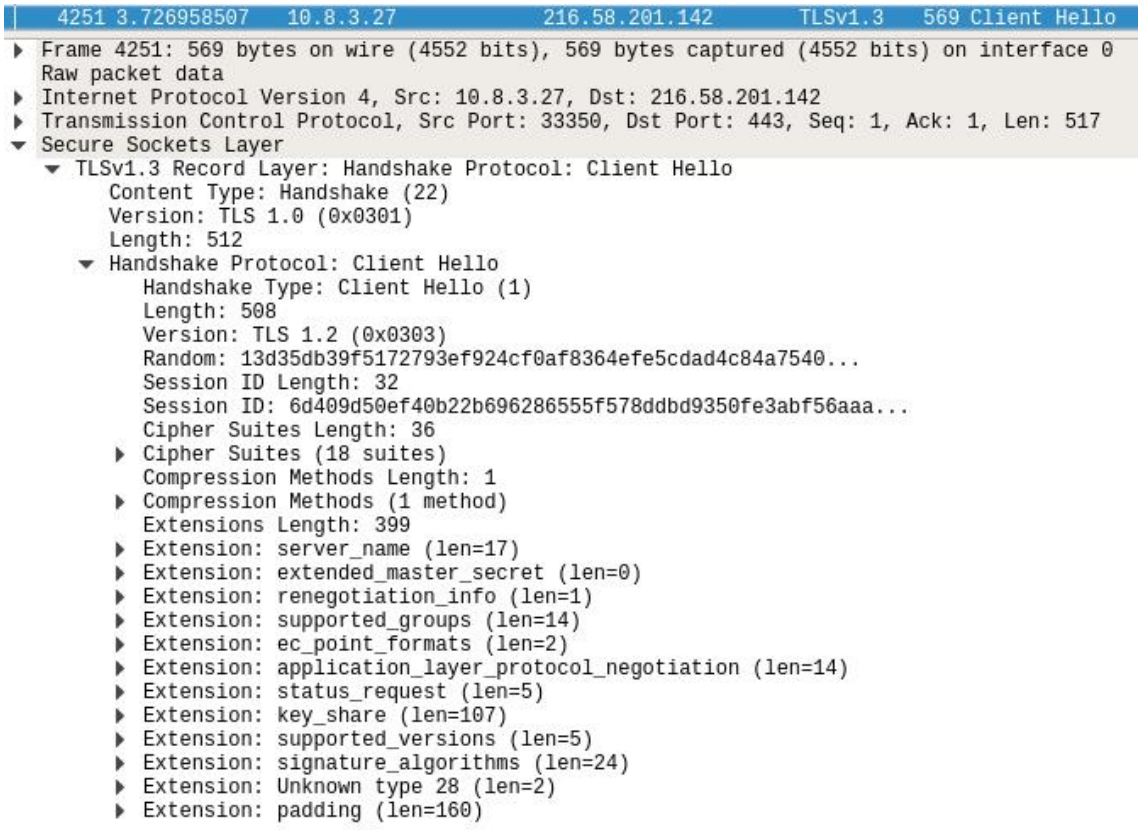
2.1.1 Criptografia

La criptografia utilitza algoritmes matemàtics per codificar la informació i donar com a resultat una cadena de bytes que no es pugui desencriptar sense un codi establert pels dos comunicadors. Aquest codi pot ser de dos tipus, la qual cosa porta a les dues categories bàsiques de la criptografia. Cal remarcar que hi ha més tipus però són derivacions de les dues anteriors:

- Criptografia simètrica: els dos comunicadors utilitzen una contrasenya per fer les funcions d'encriptar i desencriptar.

- Criptografia asimètrica: cada comunicador té dues claus, una pública que serveix per xifrar i una privada que serveix per desxifrar un missatge. Així, si l'usuari A vol enviar un missatge a l'usuari B, xifrarà la informació amb la clau pública de B de forma que l'usuari B només ho podrà desencriptar amb la seva clau privada.

En tot cas, aquesta tècnica envia el tràfic dins els camps de dades habilitats dins els protocols de comunicació, és a dir, és un flux de dades normal des del punt de vista de la xarxa. Així es pot veure a la imatge 2.02 un paquet capturat amb tràfic HTTPS, és a dir, encriptat amb una clau pública del servidor destí.



Imatge 2.02. Missatge encriptat amb HTTPS.

La criptografia no és considerada pas com una tècnica maliciosa si no més bé que dóna seguretat garantint la confidencialitat, integritat i autenticitat d'un missatge així com permet vincular tipus de fitxers amb el seu originador.

2.1.2 Tècniques d'ofuscació

Per una altra banda, la tècnica d'ofuscació serveix per amagar el protocol de xarxa mitjançant la modificació del model i contingut del tràfic que aquest genera durant una comunicació. És a dir, permet utilitzar un protocol a una comunicació de forma que per qualsevol agent intermedi que escolti la conversa es pensi que és un protocol diferent del real pel comportament que té el tràfic. Així, hi ha dues categories:

- Tràfic sense identificar: es manipula el protocol de xarxa amb mesures com l'encriptació de la capçalera de forma que no es pugui identificar el protocol.

- Tràfic impersonal: simula el comportament d'un altre protocol. Un exemple fàcil d'entendre és una xarxa P2P simulant tràfic HTTP.

Aquestes tècniques s'utilitzen per a dues raons principals: per passar tallafocs que intenten evitar tràfics de protocols concrets, com aleshores

el tràfic P2P, o per ocultar tipus de fluxos de xarxa que estan censurats com, per exemple, la xarxa Tor.

Per tant, encara que l'ofuscació pugui semblar en certa forma esteganografia en xarxa l'objectiu principal, el volum d'informació i la metodologia no són les mateixes. A l'esteganografia el propòsit és ocultar un missatge, el volum d'informació sol ser molt reduït, fins al punt de necessitar molts PDU per enviar un únic bit, i els mètodes es basen en ocultació dins un protocol de xarxa però no pas de tot el protocol.

2.2 Tipus d'esteganografia en xarxa

L'esteganografia en xarxa es basa, per tant, en utilitzar característiques dels protocols o dels fluxos de xarxa per poder introduir un missatge i transmetre'l de forma oculta. Per aconseguir-ho cal que les característiques dels protocols siguin estables i es puguin modificar però l'alteració de les quals no sigui fàcilment detectable.

D'aquesta forma, l'esteganografia se suporta en tres pilars bàsics: les comunicacions no són perfectes i es donen errors com pèrdues de PDU o retransmissions; hi ha protocols amb camps reservats per usos futurs i que actualment es poden utilitzar per transmetre bits; i que hi ha protocols que permeten interpretacions a l'hora de la seva implementació, la qual cosa es pot aprofitar per enviar informació.

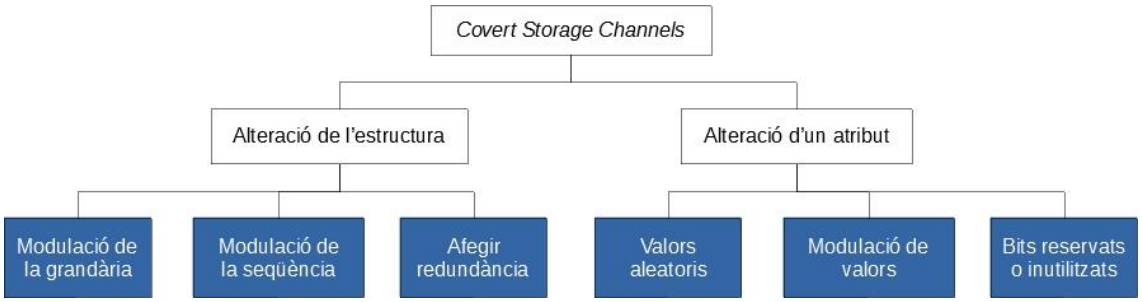
Aquestes tres característiques permeten diferenciar clarament dos tipus: les tècniques que alteren els protocols, agrupades sota el nom *storage*, i les que adapten les circumstàncies de xarxa, les quals reben el nom de *timing*. Cal destacar dins aquest darrer tipus que determinades tècniques no es veuen obligades a utilitzar un únic protocol sinó que es poden usar-ne diferents, sempre que tinguin una relació real dins una comunicació, com és aleshores el cas d'aquest projecte on s'estudia la relació entre els protocols de vídeo i àudio d'una comunicació VoLTE.

L'any 2015, Wendzel et al. varen estudiar i publicar una proposta d'organització que divideix aquests dos tipus en diferents categories [5]. Es proposen una sèrie de patrons que permeten categoritzar totes les implementacions i dissenys coneguts d'esteganografia en xarxa. Cal destacar que la comunitat científica va acollir aquesta proposta de forma molt positiva i ara mateix s'utilitza per a qualsevol nova idea de *network covert channel*.

És rellevant també explicar que hi ha dos tipus d'esteganografia en xarxa que tenen una línia molt difusa entre els dos tipus de *network covert channels* esmentats i que s'analitzaran com una categoria diferent anomenada mètodes híbrids.

2.2.1 Covert Storage Channel

Dins aquest tipus es poden englobar, com ja s'ha explicat, els *network covert channels* que o bé modifiquen l'estructura de les PDU o bé preserven l'estructura alterant en canvi parts de les capçaleres de les PDU.



Imatge 2.03. Tipus de *Covert Storage Channels*.

2.2.1.1 Modulació de la grandària

Es pot transmetre informació oculta modificant la grandària de la PDU o d'algun camp concret de la capçalera de modo que aquesta variació sigui coneguda pels dos comunicadors. Així, Murdoch i Lewis [13] modulen la grandària de paquets IP o Girling [14] aconsegueix transmetre informació alterant la longitud de la trama Ethernet.

Imatge 2.04. Modulació de la grandària.

2.2.1.2 Modulació de la seqüència

Es basa a la modificació de l'ordre dels elements dins les PDU. És un mètode fàcil d'implementar a protocols basats a estructures de capçaleres dinàmiques, com és HTTP. Una implementació que ho confirma és la proposada per Dyatlov i Castro [15] que modifiquen l'ordre de les línies a la capçalera HTTP. Una proposta basada en el protocol DHCP és la presentada per Rios et al. [16] i per FTP Zou et al. [17] utilitzen la mateixa tècnica.

2.2.1.3 Afegir redundància

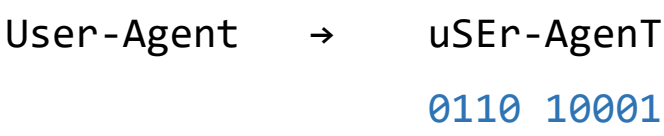
S'usa creant "espai" dins una PDU o un element concret de les PDU. És a dir, s'afegeixen bits de forma artificial que porten la informació que es vol ocultar. En aquest sentit, Dyatlov i Castro [15] varen afegir nous elements a la capçalera HTTP i Trabelshi i Jawhar [18] varen aprofitar l'opció de gravar la ruta dins la capçalera del protocol IPv4.

2.2.1.4 Valors aleatoris

Hi ha camps de certs protocols que s'inicialitzen amb valors aleatoris i, per tant, es poden utilitzar per enviar informació mitjançant un valor que mantingui l'estructura dels originals però sigui reconegut pels comunicadors. Per exemple, Bellovin [19] estableix un mètode d'estegoanàlisi per aquest tipus de *network covert channels* que utilitzen el *Initial Sequence Number* (*ISN*) per iniciar connexions TCP. Lucena et al. [20] oculta informació dins el camp MAC del protocol SSH.

2.2.1.5 Modulació de valors

Aquesta tècnica es pot implementar de moltes formes a molts protocols, però l'essència és modificar valors dels camps de la capçalera de la PDU dins rangs possibles però sense que sigui de forma aleatòria. És a dir, si hi ha un camp que pot agafar n valors diferents, consisteix a triar dos per enviar un 0 o un 1 de forma coneguda pels comunicadors. Per exemple, Lucena et al. [21] demostra que és possible implementar aquesta tècnica al camp de límit de bots del protocol IPv6. De forma similar ho fan Zander et al. [22] al TTL del protocol IPv4. La figura 2.05 mostra la implementació de Dyatlov i Castro [15] sobre la capçalera HTTP d'aquest mètode.



Imatge 2.05. Modulació de valors en HTTP.

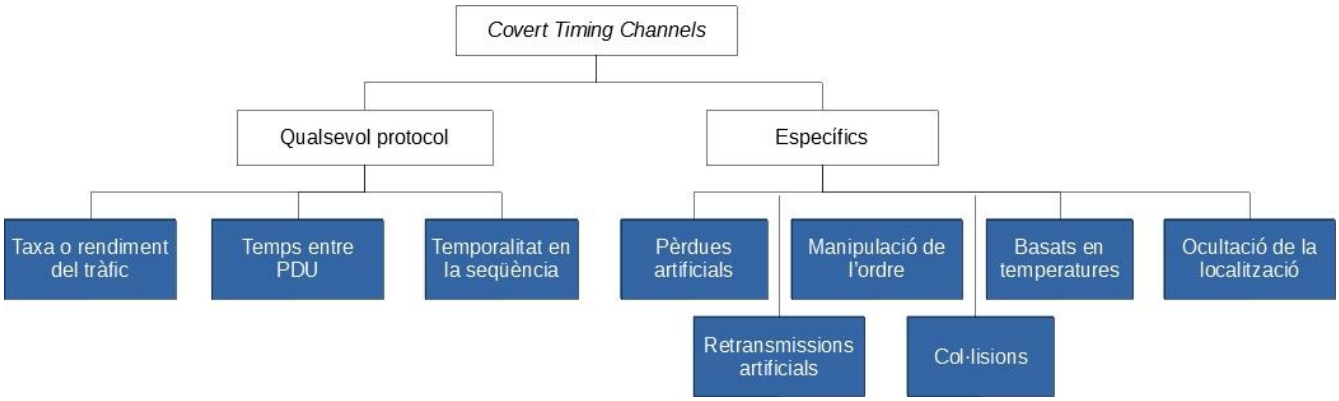
2.2.1.6 Uso de bits reservats o inutilitzats

Segons Wendzel et al. [5] aquest és el mètode més utilitzat amb 24 exemples, ja que només cal modificar aquell camp el qual no té ús real segons la definició pública del protocol en qüestió. En aquest sentit, Wolf [23] implementa aquesta tècnica sobre trames de nivell 2 i Lucena et al. [21] ho aconsegueixen sobre camps de la capçalera del protocol Ipv6.

2.2.2 Covert Timing Channel

Les tècniques que amaguen informació alterant el temps de les PDU s'agrupen en aquesta categoria. Cal explicar que solen ser tècniques que provoquen renou en la comunicació, és a dir, que no sempre permeten una recuperació del missatge de forma perfecta per retards durant l'enviament en encaminadors intermedis o per variacions a les seves cues d'encaminament. A més a més, la seva capacitat d'enviament d'informació es veu limitada enfront dels *covert storage channels*.

No obstant això, la seva detecció i/o eliminació és molt més complicada i, per una altra banda, hi ha tècniques que poden involucrar més d'un protocol diferent de forma que encara faci més complicat aplicar tècniques de contramesures.



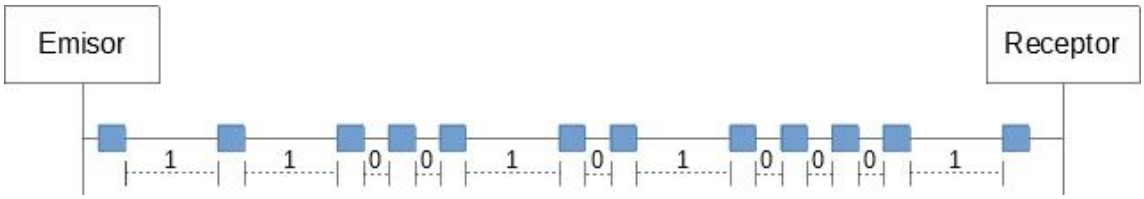
Imatge 2.06. Tipus de *covert timing channels*.

2.2.2.1 Taxa o rendiment del tràfic de xarxa

Es basa a alterar la taxa des de l'emissor de forma que el receptor obtingui el codi mesurant la taxa de les PDU que li arribin. D'aquesta forma es pot tenir un o diferents nivells de taxa d'enviament per enviar un o més bits en cada nivell. Un exemple derivat és el que van dissenyar Padlipsky et al. [24] al qual un dels nivells és el silenci, és a dir, taxa igual a 0.

2.2.2.2 Temps entre PDU

Consisteix a modificar el temps de separació entre dos PDU consecutives de forma que un o diversos llindars permetin decidir el valor binari que s'envia. Els participants han de conèixer aquests llindars de forma que puguin obtenir el missatge segons la separació temporal de les PDU que capturen. Berk et. al. [25] va demostrar que es pot utilitzar a la capa 3 del model TCP/IP, és a dir, amb els paquets del protocol IP.



Imatge 2.07. Temps entre PDU.

2.2.2.3 Temporalitat en la seqüència de les PDU

Wolf [23] va proposar alterar les operacions de gestió dels protocols per amagar missatges. L'exemple més clar són els missatges de confirmació (*acknowledgments* o ACK) els quals permeten als protocols que garanteixen l'arribada (com TCP) verificar per part de l'emissor que la informació d'una PDU ha arribat al destinatari, tal com demostra Kang et al. [26].

2.2.2.4 Pèrdues artificials

Introduint bloquejos a certes PDU es pot amagar un missatge donat un codi conegut pels extrems de la comunicació. Cal tenir present que és necessari utilitzar un protocol amb nombres de seqüència de forma que es pugui detectar la pèrdua, tal com van fer Servetto et al. a la presentació de la tècnica en [27].

2.2.2.5 Retransmissions artificials

Segons Krätzer et al. [28] és possible ocultar un missatge duplicant trames a xarxes locals sense fills. Aquest mètode proposava enviar trames duplicades de diferents connexions per codificar els valors binaris.

2.2.2.6 Manipulació de l'ordre

Es basa a alterar l'ordenació de les PDU segons el nombre identificatiu que porten dins les capçaleres. Això implica que es pot aplicar a qualsevol protocol de qualsevol capa que tingui un identificador de missatge únic i ordenat a la seva capçalera. L'exemple més clar és TCP però també es pot fer amb el protocol IP atès que té un ID únic. La proposta més interessant és la de Chakinala et al. [29] que van generar diferents models amb aquesta tècnica.

2.2.2.7 Col·lisions

Una altra tècnica del nivell 2 que aprofita el senyal de *jamming* que es genera a la xarxa quan hi ha col·lisions per evitar que se segueixi enviant tràfic. Handel i Standford [30] van proposar enviar un *jamming* a altres usuaris de forma que el receptor pugui calcular el bit segons el temps d'enviament després de l'aturada per congestió.

2.2.2.8 Basats en temperatures

Murdoch [31] va crear un canal de comunicació ocult segons les desviacions sobre el rellotge que s'introdueixen per un intermediari segons les variacions de la temperatura de la CPU que gestiona les PDU. És a dir, si una CPU ha de gestionar molta informació s'escalfarà la qual cosa provocarà desviacions temporals que permetran al receptor calcular el missatge. Poc transmetre molt poca informació (20 paquets per hora).

2.2.2.9 Temporalitat indirecta

Igual que el tipus anterior, l'emissor pot congestionar un servidor mitjançant un nombre molt elevat de peticions de forma temporal de forma que el receptor, comprovant els temps de resposta del mateix servidor, pugui obtenir el missatge. Hintz [32] va implementar una proposta d'aquest tipus.

2.2.2.10 Ocultació de la localització de l'origen

Es poden considerar comunicacions *Man-In-The-Middle* (MITM), ja que l'ocultació es fa precisament fent de *proxy* entre el receptor i un altre actor que no sap que hi ha un tercer participant que gestiona el tràfic de forma que el receptor pugui captar el missatge ocult.

2.2.3 Mètodes híbrids

Són mètodes que utilitzen dos o més de les tècniques descrites en els punts anteriors.

2.2.3.1 Pèrdua de PDU d'àudio

S'anomena LACK i es basa en el fet que el protocol RTP descarta les PDU d'àudio quan arriben molt desfasades respecte a la resta de PDU. Així, es pot modificar el contingut de la PDU perquè en lloc de portar informació de la conversació, s'insereixi informació oculta de forma que quedarà descartada pel disseny del protocol RTP però no pel receptor del missatge ocult. És per això que es considera un mètode híbrid: utilitza un retard temporal artificial però modifica el contingut del missatge. Va ser proposat

originalment per Mazurczyk et al. [33] i actualment existeixen moltes variants.

2.2.3.2 Retransmissions

És una tècnica que es pot aplicar a qualsevol protocol de qualsevol capa al qual es faci control de PDU mitjançant la confirmació pel receptor. Es coneix com a RSTEG i consisteix a obligar a l'emissor a reenviar informació ja enviada capturant els missatges de confirmació de forma que el missatge en realitat es modifica amb el contingut ocult que es desitja enviar. Es poden veure les dues components que fa considerar-lo un mètode híbrid: es modifica el flux de la xarxa i el contingut de la PDU retransmesa. Es va proposar per Mazurczyk et al. [34].

2.2.3.3 Tècniques profundes d'ocultació

Aquestes tècniques utilitzen diferents tràfics portadors o, dit d'una altra forma, diferents protocols per ocultar la informació. Això dificulta molt la detecció, ja que es podria donar que determinats protocols no s'inspeccionin i, per tant, no es detecti el mètode d'ocultació.

El que es fa en realitat és intercanviar els protocols, és a dir, intercanviar entre diferents dues o més tècniques de *network covert channels* que s'han analitzat en aquest capítol, tal com va proposar Wendzel [35].

2.3 Estegoanàlisi

L'estegoanàlisi en xarxa és el conjunt de tècniques utilitzades per tractar l'esteganografia en xarxa o, el que és el mateix, discriminar si un tràfic conté un missatge ocult segons les tècniques descrites al punt 2.2. Per tant, és important conèixer els mètodes utilitzats per a ocultar un missatge dins una conversa aparentment legítima, ja que d'una altra forma cercar tràfic amb *network covert channels* es fa molt complicat. Així, aquest coneixement és, ara per ara, una part clau per poder aplicar mesures contra l'esteganografia en xarxa.

A més a més, cal destacar que no s'ha definit encara una metodologia definida que permeti tractar a tots els tipus d'esteganografia en xarxa [9]. Les propostes actuals estan basades en mètodes concrets els quals estan dissenyats per protocols, sistemes operatius o aplicacions específiques i, per tant, la identificació i tractament dels *network covert channels* té molt de recorregut en aquest sentit.

Quan es parla de tractament de l'esteganografia en xarxa una vegada identificada, es diferencia en quatre tipus d'accions les quals no són vàlides per a tots els mètodes d'esteganografia en xarxa ni per a tots els

protocols de xarxa. No obstant això, la realitat és que les tècniques de basades en canvis estructurals dels protocols són més fàcils de tractar que les tècniques que utilitzen recursos temporals ja que es poden, fins i tot, aplicar diferents tipus de tractament.

2.3.1 Tipus de tractament

A continuació es descriuen els diferents tipus de tractaments i les possibilitats d'èxit per als tipus de tècniques d'esteganografia en xarxa descrites en aquest treball.

2.3.1.1 Prevenció

Consisteix en atacar el disseny del protocol en si mateix de forma que s'eliminin les possibilitats que porten a utilitzar els *network covert channels*. No obstant això, cercar les ambigüitats de forma manual per a tots els protocols és una feina complicada que pot portar a cometre errors i és per això que s'han intentat implementar eines automàtiques de la mateixa forma que es fa amb els *network covert channels* dins un únic equip. Donaldson et al. publiquen sobre aquesta possibilitat basant-se en el mètode SRM (*Shared Resource Matrix*) [36]. Proposen analitzar els canals de comunicació interns de cada equip tant de xarxa com entre els processos del sistema operatiu de manera que es puguin detectar possibles errors que portin a l'ús de *network covert channels*.

2.3.1.2 Eliminació

Dins aquestes tècniques hi ha dues possibilitats:

- Securització dels equips i les xarxes.

  Encara que no són tècniques pensades per acabar amb els *network covert channels*, securitzar un equip pot evitar un *programari maliciós* que utilitza aquesta forma de comunicació de la mateixa forma que prevenir l'ús de certs protocols, com ICMP, pot permetre l'eliminació de determinats tipus d'esteganografia en xarxa de qualsevol dels tipus.

- Normalització del tràfic de xarxa.

  Aquesta tècnica sí que es considera específica contra els *network covert channels*, ja que la idea és normalitzar bé les capçaleres de forma que, per exemple, camps sense utilitzar sempre tinguin el mateix valor o camps desconeguts es facin desaparèixer, o bé els fluxos de xarxa reordenant, per exemple, les PDU pel seu nombre de seqüència. Aquesta tècnica s'ha estudiat amb més profunditat i fins i tot la literatura distingeix entre dos tipus: *stateless*, que revisen

cada PDU de forma independent, i *stateful*, que avaluen les PDU com converses però a canvi requereixen memòries més grans [21].

2.3.1.3 Limitació

Es basen a limitar la capacitat d'enviar informació del *network covert channel* i estan orientats als de tipus *timing*. Hi ha diferents possibilitats com introduir renou, forçar les grandàries de les PDU o la taxa de transferència o limitar la relació entre dos tipus de protocols diferents [37]. En qualsevol cas, són tècniques fetes a mesura contra un determinat mètode d'esteganografia en xarxa la qual cosa permet que petites variacions podrien no ser limitades perquè no es detectin.

2.3.1.4 Detecció

La detecció de l'esteganografia en xarxa és l'objectiu d'aquest treball i per això s'ha vist oportú deixar el tercer capítol sencer per a explicar els diferents mètodes que existeixen i les seves particularitats.

2.3.2 Tècniques actuals i futures

Tal com s'ha explicat, realment no existeixen productes al mercat que puguin servir com a mètode preventiu per evitar que a una xarxa local s'utilitzin *network covert channels* per comunicar-se amb l'exterior. No obstant això, hi ha productes anomenats *Data Leakage Protection* (DLP) que implementen alguns algoritmes de detecció però amb uns resultats dubtosos [11].

Les propostes actuals dels investigadors són:

- Mètodes per resoldre problemes concrets: és a dir, identificat un mètode per ocultar informació, cercar la manera d'aplicar un tractament dels descrits.

- Mètodes generalistes per a diferents tipus d'esteganografia en xarxa: una idea prometedora de difícil implementació per la diferent naturalessa dels *network covert channels* i dels protocols de xarxa. No obstant això, Wendzel et al. [10] van publicar un mètode basat en una puntuació per detectar tràfic amb missatges ocults per a diferents mètodes d'esteganografia en xarxa.

A les publicacions esmentades es reflecteix que, encara que la prevenció, eliminació i limitació són una possibilitat, la detecció mitjançant la intel·ligència artificial intentant agrupar diferents tipus d'esteganografia en xarxa és l'evolució de les tècniques actuals. El primer pas seria agrupar crear mètodes de detecció de les tècniques esmentades, en lloc de fer-ho per a cada tècnica d'esteganografia concreta. La següent passa seria

l'agrupació de tècniques per finalitzar amb el conjunt d'un tipus concret o del màxim de tècniques.

3. Mètodes de detecció

A la literatura es presenten dos mètodes generals: anàlisi estadístics o aprenentatge automàtic amb classificadors supervisats. En aquest capítol es presenten els dos tipus i subtipus, amb exemples d'utilització a l'esteganografia en xarxa.

El procés de detectar *network covert channel* es fa de la mateixa forma: es cerca comparar tràfic legítim amb tràfic que porta un missatge ocult de forma que la diferència de comportaments permeti treure conclusions. El tràfic es compara en forma de característiques o *features* i s'estableix un llindar per a decidir si el tràfic porta o no un missatge ocult.

Pel que fa a l'anàlisi estadístic, el llindar ha de ser pensat i especificat per una persona durant l'entrenament però la pròpia comunitat ha assumit que no està clar com determinar aquest límit. Per una altra banda, l'aprenentatge automàtic determina aquest llindar el mateix algorisme durant la fase d'entrenament. Cal tenir present que també, a vegades, és molt complicat interpretar el resultat.

Aquest capítol s'ha dividit en dos punts encara que el principal atractiu pel projecte és la segona part on s'expliquen els models d'aprenentatge automàtic o *machine learning*.

3.1 Mètodes estadístics

Hi ha nombroses tècniques basades en estadístiques que s'han utilitzat per a detectar *network covert channels* però també hi ha mètodes de *machine learning* que utilitzen els resultats dels mètodes estadístics com *features* per a entrenar el classificador.

En aquest projecte es presenten dos dels més utilitzats amb les motivacions descrites, els tests KS i KLD. Són tots dos tests que han servit per a detectar però també per demostrar la indetectabilitat de mètodes d'esteganografia en xarxa com a l'estudi esmentat de Zhang et al. [6].

3.1.1 Test Kolmogorov-Smirnov (KS)

Permet comparar dues mostres per verificar si pertanyen a la mateixa distribució. Com més petit sigui el resultat, més semblant són, mentre que un valor elevat significa que són diferents. Una particularitat d'aquest mètode és que és vàlid per qualsevol mena de dades.

Si es considera $F(x)$ la funció empírica de distribució d'una única dimensió, el test KS per a dues funcions és:

$$D_{KS} = sup_x |F_1(x) - F_2(x)|$$

On sup_x és la distància més allunyada.

Com a exemple, Zander [38] va utilitzar aquest test per generar *features* per entrenar un algorisme d'aprenentatge automàtic amb distribucions de tràfic tant normal com amb un missatge ocult utilitzant la tècnica per encobrir de temps entre PDU.

Zhang et al. [6] utilitzen aquest mètode per verificar que no es pot detectar els dos mètodes de *network covert channels* basats a alterar l'ordre dels missatges.

3.1.2 Test Kullback-Leibler Distance/Divergence (KLD)

Aquest mètode és una mesura no simètrica de la diferència entre dues funcions de probabilitat. En aquest cas, el resultat sempre és igual o superior a zero i com més gran, més diferents són les funcions de probabilitat.

Donades dues funcions de probabilitat $P = \{p_{1,\dots}, p_n\}$ i $Q = \{q_{1,\dots}, q_n\}$, la distància, divergència o entropia relativa de *Q* amb respecte de *P* [39] es calcula com:

$$D_{KL}(P, Q) = \sum_i p_i \, ln \frac{p_i}{q_i}$$

De la fórmula es pot veure perquè es diu que és no simètrica, la divisió provoca que la inversió de l'ordre pugui donar resultats diferents, i què mesura aquest test a un conjunt discret, el resultat és la diferència en el nombre de valors que pertanyen al mateix grup. Cal remarcar que s'utilitza la suma acumulativa al llarg de la variable per calcular-la.

El test KLD el van utilitzar Zhai et al. [40] per a detectar missatges ocults en alteracions dels *flags* de la capçalera dels segments TCP. Varen aconseguir resultats bons quan el volum de segments alterats era elevat enfront del nombre de segments sense alteració.

De la mateixa forma que el test KS, Zhang et al. [6] van usar el test KLD per verificar que no es pot detectar els dos mètodes de *network covert channels* basats a alterar l'ordre dels missatges.

3.2 *Machine Learning*

La intel·ligència artificial no és concepte del segle XXI ni tan sols del segle XX. Encara que definit d'altres maneres, filòsofs i científics de segles anteriors ja havien somiat amb idees semblants, com per exemple Ramón Llull al segle XIII que va plantejar la idea de que un maquinar artificial pogués raonar [41]. No obstant això, no va ser fins a l'any 1950 que es va començar a materialitzar de la mà d'Alan Turing a l'article *Computing Machinery and Intelligence* [42], al qual va proposar la seva famosa prova per resoldre si una màquina era intel·ligent anomenada "Test de Turing". En 1956 el terme "intel·ligència artificial" es va admetre al congrés de Darmouth com definitòria per les accions que portessin a crear programari intel·ligent.

Per aconseguir que una màquina pugui raonar o prendre decisions per si mateixa, cal que hagi après de situacions que li permetin arribar a conclusions concretes. Aquest procés es defineix *machine learning* i s'aconsegueix mitjançant una sèrie d'algorismes basats a desenvolupaments algebraics. Entendre amb profunditat la complexitat d'aquests algorismes no és objectiu d'aquest projecte però es farà una breu descripció dels principals mètodes, els quals es poden categoritzar en els següents tipus principals:

- Supervisat: es basa a entrenar l'algorisme amb uns conjunts de dades prèviament determinats perquè doni com a resultat una funció que separi de forma adient els grups. Segons l'algorisme, el resultat pot ser un nombre o un dels grups que han servit per entrenar però l'objectiu és sempre el mateix, la predicció.

- No supervisat: no requereix un entrenament sinó que el mateix algorisme cerca les semblances i diferències a un únic conjunt de dades. Així, l'objectiu d'aquests mètodes és l'agrupació de les dades introduïdes.

- Semisupervisat: s'utilitzen dades prèviament classificades i dades que no ho estan de forma que les primeres ajudin a classificar les segones, el qual és l'objectiu principal d'aquest tipus de *machine learning*.

- Per reforç: l'aprenentatge es fa mitjançant la maximització d'una recompensa que s'alimenta amb un reforçament. És a dir, es basa a cercar les accions que permetin arribar de la millor forma possible a un resultat concret.

En aquest projecte, atès que hi ha dos conjunts de dades diferenciats des del punt de vista de la motivació (un tràfic normal i un altre amb un missatge ocult) i que l'objectiu és obtenir una funció per resoldre si nous tràfics de xarxa contenen o no missatges amagats, el tipus de *machine learning* escollit per assolir l'objectiu del treball és el supervisat orientat a la classificació.

3.2.1 SVM

Support Vector Machines o SVM és un model d'aprenentatge automàtic supervisat que es pot utilitzar en anàlisi de classificació o de regressió, essent els primers tipus el que resulta més interessant per al projecte tal com s'ha explicat.

3.2.1.1 Característiques

S'ha de tenir en compte que:

- És molt efectiu en espais de grans dimensions, encara que siguin moltes més del nombre de mostres.
- Pot haver-hi sobre-ajustament si el nombre de *features* és gran.
- Empra poca memòria perquè utilitza el conjunt de dades més a prop dels marges (anomenats *support vectors*).
- No estima directament la probabilitat, cal usar altres mètodes que poden resultar costosos.
- Molt versàtil donat la varietat de funcions *kernel*, fins al punt que es poden implementar-ne de pròpies.

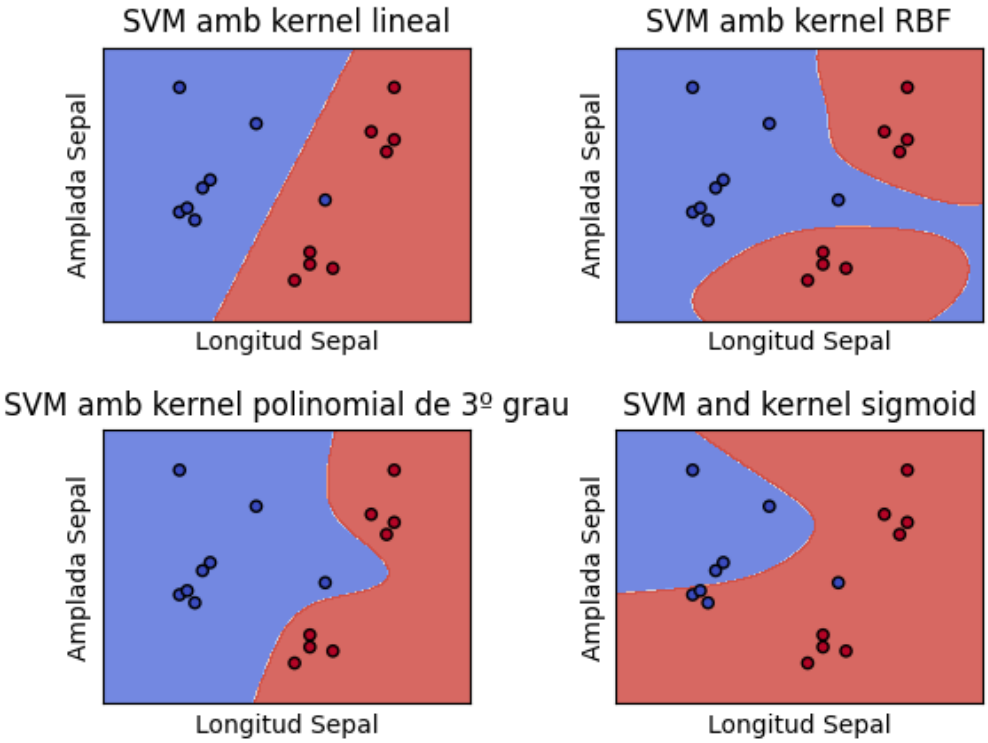
3.2.1.2 Tipus de SVM

Segons el resultat, SVM es pot classificar en lineals i no lineals. Els SVM lineals donen com a resultat una línia o pla recte o un hiperplà N-dimensional però no té moltes aplicacions reals, ja que poques vegades es pot separar adientment dos grups d'un conjunt de dades de forma simple.

Els SVM no lineals són molt complexos perquè el resultat és qualsevol forma geomètrica diferent d'una línia o pla recto. És evident que la

computació exigeix molt més en aquests tipus de models de *machine learning* i per això s'utilitzen uns nuclis o *kernels* que transformen les dades en un espai de característiques de dimensió més elevada de manera que es puguin analitzar la situació com un problema lineal. Hi ha diferents tipus de *kernels* els quals usen distints desenvolupaments matemàtics:

- Polinomial-homogènia.
- Perceptron.
- Funció de base radial Gaussiana.
- Sigmoide.



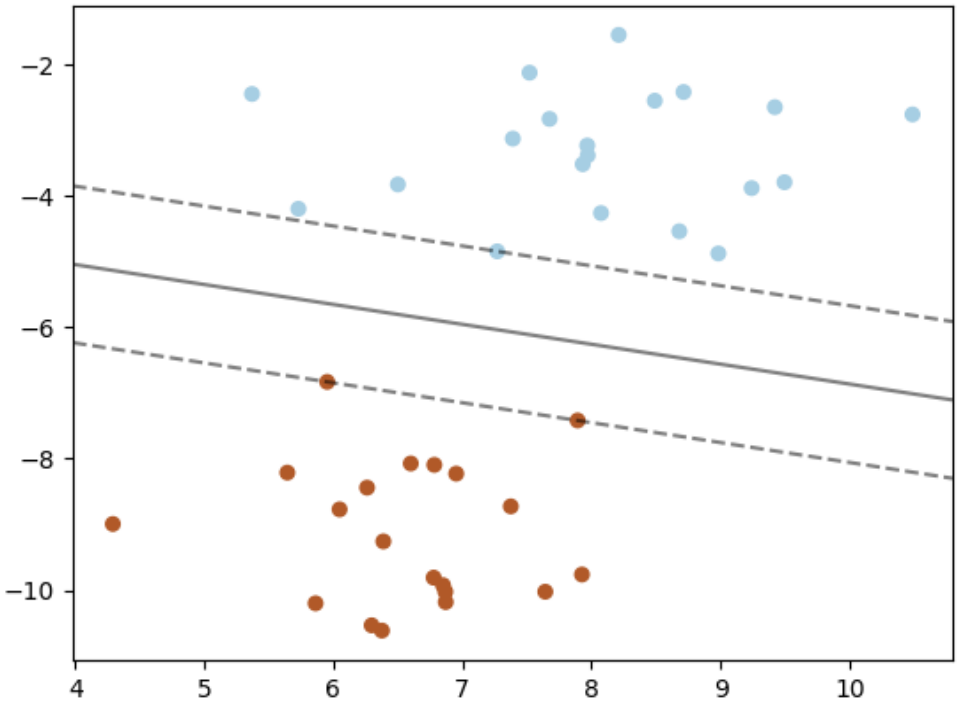
Imatge 3.01. Diferents *kernels*.

3.2.1.3 Breu introducció matemàtica.

SVM requereix per entrenar un conjunt de dades que ha d'estar etiquetat en dos grups:

$$(x_1, y_1), \ldots, (x_n, y_n)$$

On $x_i \in \mathbb{R}^d$ són les variables que es volen analitzar i $y_i \in \{-1,1\}$ és l'etiqueta que defineix el grup. SVM cerca un hiperplà que separi els dos grups deixant un marge o distància igual entre els dos grups. L'hiperplà que optimitzi aquest marge, serà el resultat final.



Imatge 3.02. Marge al voltant de l'hiperplà.

Donat qualsevol punt, la distància a l'hiperplà queda reflectida per:

$$w'x + w_0 = 0$$

Si es considera $\hat{x}$ el punt de l'hiperplà més proper a x, la distància entre ells es calcula com:

$$d = \frac{|w'x + w_0|}{\|w\|}$$

I el marge queda definit per la mínima distància possible:

$$\min_i \frac{y_i(w'x + w_0)}{\|w\|} = \frac{1}{\|w\|}$$

Per tant, el que es vol aconseguir és la maximització del marge per a tot el conjunt de dades, la qual cosa es resol amb el següent problema:

minimitzar $\quad \|w\|^2/2$

s. a. $y_i(w'x_i + w_0) \geqslant 1, \qquad i = 1, \ldots, n$

Per aquest projecte cal tenir present que les dades no tenen per què separar-se linealment com a la figura 3.02 i, per tant, cal introduir dues variables:

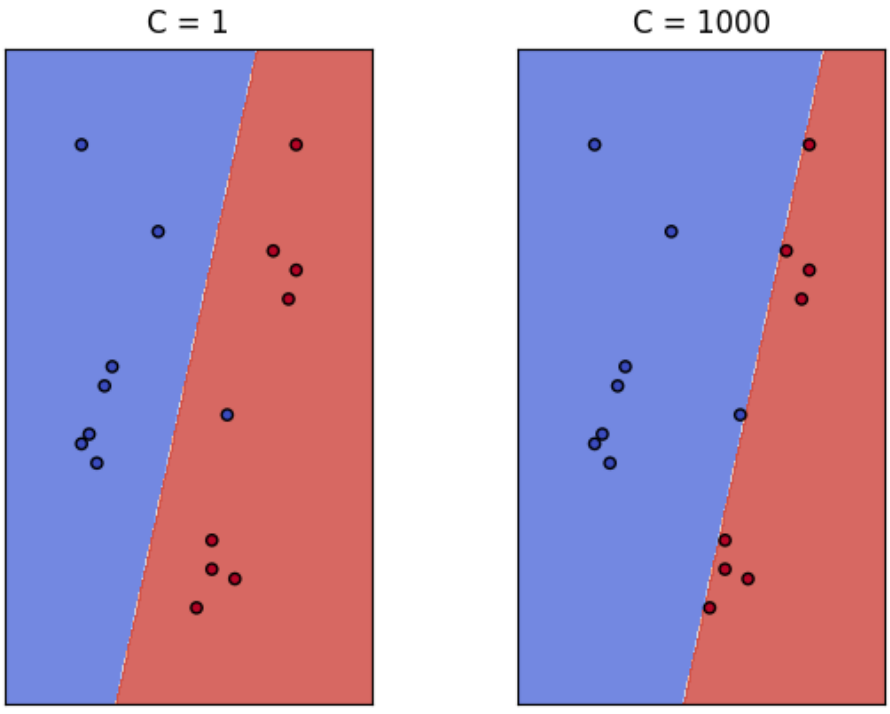
- ξ : variables que permeten relaxar les restriccions per permetre errors de classificació.

- *C* : l'usuari selecciona aquest valor per determinar la penalització dels errors.

Així, el problema queda de la següent forma:

minimitzar $\|w\|^2/2 + C\sum_{i=1}^{n}\xi_i$
s. a. $y_i(w'x_i + w_0) + \xi_i \geqslant 1, \qquad i = 1$
$\xi_i \geqslant 0, \qquad i = 1$

D'aquesta forma, segons el valor de *C*, l'hiperplà presentat pot ser totalment diferent tal com es pot veure a la figura 3.03.



Imatge 3.03. Diferents valors de *C*.

L'hiperplà donat com a resultat final és un vector amb els pesos de les *features* que s'utilitza com un classificador, ja que al combinar-lo amb dades, dóna un valor que representa la confiança que les dades siguin d'un determinat grup.

3.2.1.4 Exemples a l'estegoanàlisi en xarxa.

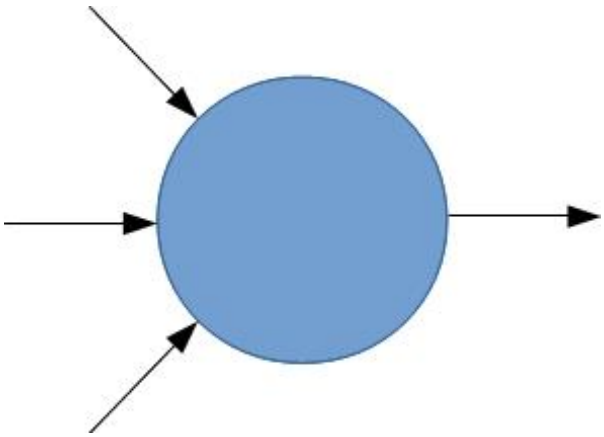
Pel cas de l'estegoanàlisi en xarxa, hi ha molts casos d'èxit amb models SVM. Així, Sohn et al. [43] va demostrar que es podien *detectar network covert channels* dins el camp ID de la capçalera dels paquets IP o del camp del número de seqüència inicial de la capçalera dels segments TCP. Amb diferents conjunts de *features* van aconseguir precisions de detecció del 99%.

Archibald i Ghosal [44] varen aconseguir precisions per sobre el 90% amb un classificador SVM detectant tràfic alterat utilitzant la tècnica de modificar la temporalitat entre les PDU. Els resultats eren bons amb finestres de temps molt llargues.

Amb SVM també s'ha intentat generalitzar el model per a diferents tipus de *network covert channels*. Shen et al. [45] ho varen aconseguir per deu tipus diferents basats a deu camps diferents de les capçaleres TCP/IP mitjançant l'extracció de les *features* amb la correlació dels valors de les capçaleres.

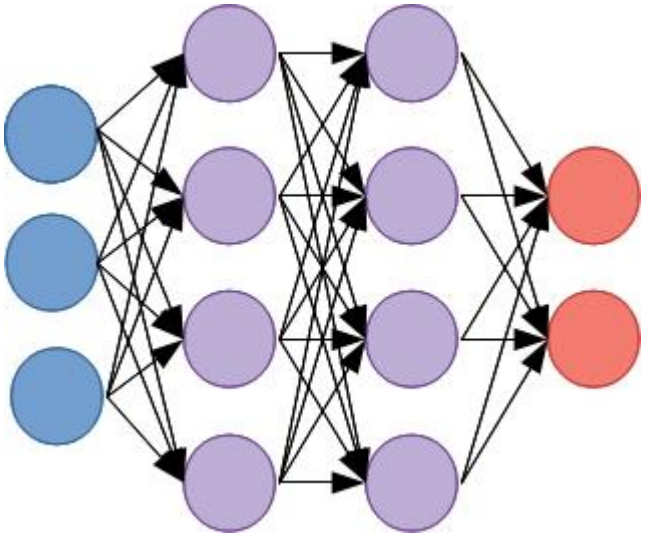
3.2.2 Xarxes neuronals

Aquest model es basa a una emulació del funcionament del cervell, el qual passa informació entre les neurones mitjançant un procés anomenat sinapsis i les ramificacions que les uneix entre elles. De la mateixa forma, es modelen neurones artificials com un node amb un nombre indefinit de senyals d'entrada i una única sortida, la qual cosa simula la forma i ramificacions. Cada entrada al node de la neurona es caracteritza per un pes que representa la intensitat de les connexions sinàptiques que té cadascuna.



Imatge 3.04. Neurona artificial.

Les xarxes neuronals utilitzen diferents capes on cada neurona de la capa N es connecta amb les sortides de totes les neurones de la capa anterior i la seva sortida serveix d'entrada a totes les neurones de la següent capa. Encara que evident, cal destacar que la primera capa no té entrades i la sortida de la darrera és el resultat final.



Imatge 3.05. Xarxa neuronal.

3.2.2.1 Característiques

És important remarcar que:

- Permet aprendre en models no lineals.
- Si existeixen més d'un mínim local, les capes intermèdies poden donar resultats inesperats.
- Es pot utilitzar per aprendre models en temps real.
- Requereix molt treball preparant la xarxa.
- És sensible a l'escalat de *features*.

3.2.2.2 Tipus de xarxes neuronals

Les xarxes neuronals es poden dividir segons diferents criteris entre els quals estan la topologia, el tipus d'aprenentatge o la funcionalitat però la categorització més reconeguda a la literatura és segons les connexions que s'estableixen entre les diferents capes de neurones que formen la xarxa[46].

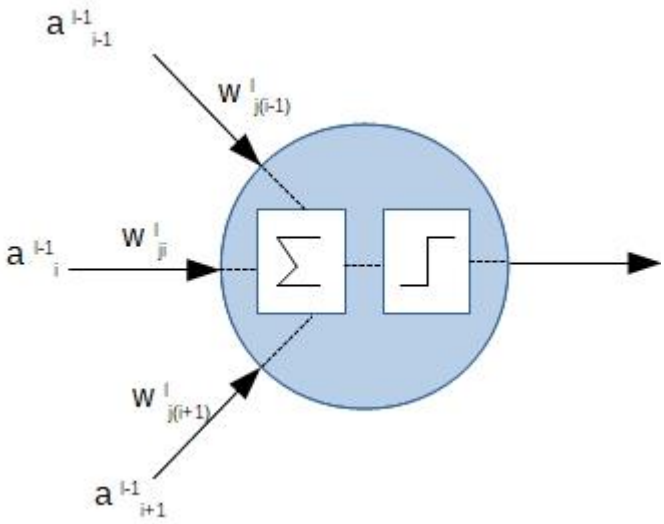
- Amb connexions cap endavant (*feedforward*): els pesos fan que la informació es propagui cap endavant, és a dir, no hi ha connexions laterals ni cap enrere com a les xarxes neuronals convolucionals.

- Amb connexions cap endavant i cap enrere (*feedforward/feedback*): hi ha enllaços en els dos sentits en cada capa i, per tant, pesos a cada enllaç. Se solen anomenar *Recurrent Neural Network* (RNN).

3.2.2.3 Breu introducció matemàtica.

Els dos tipus de xarxes neuronals tenen desenvolupament matemàtics diferents però l'essència és la mateixa. La sortida de cada neurona és la suma de les entrades multiplicades pel seu pes amb un filtre que limita el valor de sortida:

$$a_j^l = f\left(\sum_i w_{ji}^l a(l-1)_i + b_j^l\right)$$



Imatge 3.06. Esquema matemàtic d'una neurona artificial.

On *a* representa el valor de sortida d'una neurona artificial, *i* i *j* són els nivells dins cada capa, *l* és el nombre de la capa on es troba la neurona, *w* és el pes assignat a un enllaç i *b* és un valor que no sempre està present i determina la predisposició a treure un valor o un altre amb independència del pes.

El filtre f és una funció anomenada "d'activació" perquè permet adaptar la sortida a valors acotats segons les necessitats. Cal destacar que es poden implementar xarxes neuronals amb capes amb diferents funcions

d'activació i, a xarxes complexes, els nodes poden tenir diferents funcions d'activació. Les principals funcions d'activació són:

- Funció identitat.

$$f(x) = x$$

- Funció escaló.

$$f(x) = \begin{cases} 0 si x < 0 \\ 1 si x \geqslant 1 \end{cases}$$

- Funció Sigmoide.

$$f(x) = \frac{1}{1+e^{-x}}$$

- Funció Rectificadora o *ReLU*.

$$f(x) = max\{0, x\} \forall x \geqslant 0$$

- Funció gaussiana.

$$f(x) = e^{\frac{-x^2}{2}}$$

- Funció tangent hiperbòlica.

$$f(x) = \frac{1-e^{-2}x}{1+e^{-2}x}$$

L'aprenentatge de les xarxes neuronals són el que fa més atractiu aquest model de *machine learning* i el mètode més comú es basa al que es diu aprenentatge per correcció d'errors i la seva implementació es coneix com a *backpropagation*. El que es fa és entrenar la xarxa sobre una sortida desitjada (t_k) de forma que la sortida es compara amb la sortida desitjada i es calcula un error per a cada neurona que es propaga cap enrere.

$$E = \frac{1}{2}\sum_{k=0}^{N}(t_k - a_k^M)$$

On *N és* el nombre de nivells de la darrera capa i *M* el nombre de capes (és a dir, el valor final de sortida). Aquest error es propaga a tots els pesos de forma proporcional a la contribució de cada enllaç al resultat final segons la següent fórmula:

$$\Delta w_l = -\eta \frac{\partial E}{\partial w_l} = \eta \sum_k a_k^l \, (t_k - a_k^M)$$

3.2.2.4 Exemples a l'estegoanàlisi en xarxa.

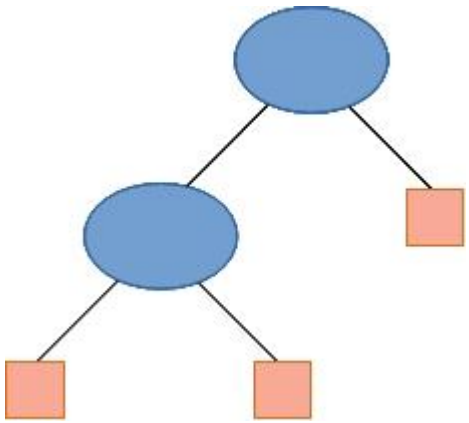
Encara que no s'ha utilitzat tant, també es troben detectors que funcionen amb aquest model. Tumoian i Anikeev [47] van entrenar una xarxa neuronal per detectar *network covert channels* sobre el camp ISN (*Initial Sequence Number*) de la capçalera dels segments TCP. Varen aconseguir resultats de precisió del 99% entrenant al classificador a predir els successius ISN per a diferents sistemes operatius.

Sun et al. [48] utilitzen una RNN optimitzada amb el model LSTM per aconseguir una precissió del 91% encara que no es detalla sobre quins tipus de *network covert channels* ho varen aplicar.

3.2.3 Arbres de decisió

Els arbres de decisió no són un model exclusiu de l'aprenentatge automàtic. En realitat, el *machine learning* s'ha apropiat d'aquesta tècnica per a treure conclusions a causa de les seves característiques que ja s'aprofitaven a l'hora de prendre decisions en estadística o en mineria de dades.

Els arbres de decisions representen els valors en nodes que poden ser de decisió o de resposta. Els primers s'associen amb els atributs i representen les decisions, ja que d'ells surten tantes branques com possibles valors prenen els atributs associats. Els nodes de resposta estan associats a les classificacions que es proporcionen durant el recorregut de l'arbre.



Imatge 3.07. Arbre de decisió.

3.2.3.1 Característiques

Els factors de decisió per triar aquest model enfront d'altres han de tenir en compte els següents punts:

- És fàcil d'entendre i d'interpretar tant el recorregut com el resultat amb funcions booleanes bàsiques (AND i NOT).
- Es complica el raonament amb altres funcions booleanes (com XOR).
- Requereix que totes les possibilitats (branques) tinguin dades.
- Poden gestionar tant dades numèriques com categories.
- Petites variacions poden portar a inestabilitats dins l'arbre.
- Poden gestionar problemes amb múltiples sortides.
- El cost és logarítmic segons el nombre de dades utilitzades per entrenar l'arbre.
- És conegut que el problema d'aprendre la solució òptima és NP-complet per a conceptes simples però això porta a solucions locals i, per tant, no sempre es pot garantir la millor solució global.
- És possible validar el model usant tests estadístics.

3.2.3.2 Tipus d'arbres de decisió.

Bàsicament hi ha dos tipus basats en el tipus de sortida:

- Regressió: si la sortida es pot considerar un nombre real, és a dir, un valor dins un conjunt continuo.
- Classificació: si la sortida està acotada a dues o més possibilitats o, dit d'una altra forma, a valors discrets.

Aquesta diferència es veu reflectida als algorismes que implementaven els arbres de decisió. Així, per variables contínues el més conegut es diu *AdaBoost*, que també es pot utilitzar amb variables discretes, mentre que per conjunts discrets el més popular és C4.5. En aquest projecte s'utilitzen l'anomenat CART (*Classification And Regression Tree*) que és una versió del C4.5 i l'*AdaBoost*.

3.2.3.3 Breu introducció matemàtica

Com a la resta de models, partim d'un conjunt de dades etiquetes per a entrenar el classificador.

$$(x_1, y_1), \ldots, (x_n, y_n)(x_1, y_1), \ldots, (x_n, y_n)$$

On $x_i \in \mathbb{R}^d$ són les variables a analitzar i $y_i \in \{-1,1\}$ és l'etiqueta.

Per simplificar aquesta introducció es considera que de cada node de decisió només surten dues possibilitats representades amb els nombres 0 i 1. Per tant, per al node *m* representat per *Q*, cada branca es divideix en $Q_0(\theta)$ i $Q_1(\theta)$ on $\theta = (j, t_m)$ és el factor de decisió, és a dir, segons la característica *j* i el valor límit t_m. Així, la decisió es calcula com:

$$Q_0(\theta) = (x, y)|x_j \leqslant t_m$$
$$Q_1(\theta) = Q \setminus Q_0(\theta)$$

Per a decidir quin dels dos camins s'ha de prendre es defineix un concepte de puresa $H\big(Q(\theta)\big)$ per a cada branca de sortida de forma que el resultat es presenta com un guany:

$$G(Q, \theta) = \frac{n_0}{N_m} H\big(Q_0(\theta)\big) + \frac{n_1}{N_m} H\big(Q_1(\theta)\big)$$

On N_m és la profunditat a la qual es troba aquest node dins l'arbre. L'objectiu és seleccionar el paràmetre que minimitza la impuresa, és a dir:

$$\theta^* = argcmin_\theta G(Q, \theta)$$

Es segueixen els camins $Q_0\big(\theta^*\big)$ i $Q_1\big(\theta^*\big)$ recursivament fins que s'arriba a la màxima profunditat.

Cal destacar per desenvolupar un algorisme de classificació, la funció per definir la impuresa pot ser de diferents tipus, dins les quals les més comunes són:

- *Gini*

$$H\big(Q(\theta)\big) = \sum_k p_{mk}\,(1 - p_{mk})$$

- Entropia

$$H\big(Q(\theta)\big) = -\sum_k p_{mk}\, log(p_{mk})$$

- *Misclassification*

$$H\big(Q(\theta)\big) = 1 - max(p_{mk})$$

On p_{mk} és la proporció de la classe *k* (és a dir, valors de *y* que en aquest projecte només hi ha dues) i es considera de la següent forma:

$$p_{mk} = \frac{1}{N_m} \sum_{x_i} I\,(y_i = k)$$

3.2.3.4 Exemples a l'estegoanàlisi en xarxa.

Zander [38] va utilitzar un classificador C4.5 per a detectar canals encoberts al camp TTL dels paquets de la capçalera IP. Va veure que com més a prop de l'origen, millor però que fins i tot a prop del destí la precisió era del 95% amb molts paquets. Amb pocs, la precisió baixava al 85-90%.

Wendzel i Zander [49] varen mostrar que és possible detectar *network covert channels* que utilitzen la tècnica avançada d'intercanvis de protocols amb un classificador C4.5. El classificador es basa a dues mètriques concretes, el nombre mitjà de canvis per paquet i la mitja del temps entre els intercanvis de protocols. Els resultats són del 98-99% amb molts intercanvis però es redueix fins al 96% amb menys intercanvis.

Wendzel [7] va utilitzar una altra vegada el classificador C4.5 per a detectar una comunicació encoberta basada en l'ordre dels missatges. El resultat és que la precisió és superior al 99,5% amb un ràtio de falsos positius inferior a l'1% si hi ha tres o més PDU en ús. Si s'utilitzen dos PDU, la precisió és del 94,5% i el nombre de falsos positius és del 5,19%.

3.3 Mètriques

Per finalitzar, cal definir com es mesuren els resultats dels algorismes. Per això es defineixen les mètriques que són les representacions dels resultats de l'execució d'un algorisme d'aprenentatge automàtic. Triar la més adient per a cada situació és un aspecte clau per avaluar correctament el resultat.

Les representacions es fan en funció de la anomenada *matriu de confusió* que agrupa les quatre possibilitats de classificació:

True Positive (TP)	*False Positive (FP)*
False Negative (FN)	*True Negative (TN)*

Imatge 3.06. Matriu de confusió.

En relació amb el projecte cada categoria es pot definir com:

- *True Positive* (TP). Si el resultat es correspon amb la realitat. Per exemple, si un conjunt de dades es categoritza com *network covert channel* i és part d'un tràfic que porta un missatge ocult.

- *False Positive* (FP). Si el resultat s'ha considerat d'una categoria a la qual no pertany. En aquest cas seria si es considera que un tràfic

altera deliberadament el flux per incloure un missatge però no és cert.

- *False Negative* (FN). Si el resultat no es considera del tipus que és. És a dir, si no es categoritza com *network covert channel* però sí que és un tràfic que forma part d'un canal encobert.

- *True Negative* (TN). Si el resultat mostra que no s'ha considerat d'una categoria i realment no ho és. Així, un tràfic que no porta un missatge ocult i no és realment un missatge ocult.

Les diferents mètriques combinen aquests valors d'una forma particular perquè tenen un significat propi. Les mètriques de classificació més interessants pel treball són:

- *Accuracy*.

  Mesura la quantitat total d'elements que s'han classificat correctament.

  $$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

  D'aquesta forma es veuen quants casos s'han encertat sobre el total de resultats i per això cal la divisió pel nombre complet d'elements avaluats. El resultat és un percentatge.

- Precisió.

  Defineix el nombre d'elements correctament identificats com positius de tots els que es varen considerar positius.

  $$Precisió = \frac{TP}{TP + FP}$$

  Es relacionen els positius que són correctes i els que no ho són (FP). Un valor elevat com a resultat demostra que hi ha massa negatius que es consideren possitius

- Sensibilitat o *recall*.

  Mesura el nombre total de positius detectats sobre el nombre total de positius que hi havia al conjunt de dades.

  $$Sensitibilitat = \frac{TP}{TP + FN}$$

  Els que són positius però no han estat detectats es comptabilitzen als falsos negatius (FN). Com més petit sigui FN, més positius s'hauran detectat i més gran serà aquest resultat.

- *F1-score*.

  Relaciona les mètriques *Precisió* i *Sensibilitat* de la següent forma:

  $$F1 = \frac{2\ x\ Precisió\ x\ Sensibilitat}{Precisió + Sensibilitat} = \frac{2\ *TP}{2*TP+FP+FN}$$

  Es veu a la fórmula resultant que només es relacionen els conceptes que han de veure amb els positius però no es valoren els negatius.

- *MCC*.

  Relaciona tots els valors de la matriu de confusió i es considera un resultat balancejat entre els positius i els negatius.

  $$MCC = \frac{TP x\ TN - \ FP\ x\ FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

  Com més falsos positius o falsos negatius hi hagi, pitjor, però cal tenir present que els resultats es mouen entre *-1* i *+1* de forma que els valors més petits indiquen més quantitat d'errors de classificació.

Com s'ha vist cada mètrica proporciona una informació que no té per què resultar adient a tots els escenaris. Per explicar-ho s'utilitza l'exemple d'un detector d'una malaltia. Es pot imaginar que aquesta malaltia afecti el 5% de la població de forma que si l'algorisme dóna sempre com a resultat que el pacient no la té, la mètrica *accuracy* serà un 95%. La realitat és que aquesta mètrica no serveix en aquest cas.

En aquest treball l'escenari és molt semblant però encara més extrem. Encara que no s'han trobat estudis amb percentatges, és fàcil pensar que el volum total de tràfic que es mou amb missatges ocults per tècniques d'esteganografia en xarxa sigui realment molt petit.

A la literatura exposada en aquest capítol, les mètriques més utilitzades són la precisió i el *F1-score*. En aquest projecte es presenten els resultats amb les mètriques *F1-score* i MCC, atès que és molt més completa que les anteriors en tenir en compte també els resultats TN.

4. Eines desenvolupades.

Al llarg de les següents pàgines es descriu tot el desenvolupament fet per assolir els objectius descrits a la introducció. Es presenten tant un mètode d'esteganografia en xarxa basat en l'alteració de l'ordre de les PDU com l'eina que permet detectar-ho.

L'objectiu principal d'aquest TFM és desenvolupar amb èxit un detector de *network covert timing channels* de la tècnica "manipulació de l'ordre" (vegeu 2.2.2.6) basat en *machine learning*. Per aconseguir-ho, hi havia tres problemes per resoldre: l'obtenció de tràfic normal d'un protocol però també tràfic del mateix protocol amb un missatge ocult, que el volum de tràfic fos suficient per entrenar un classificador i la implementació del classificador.

Com que la idea inicial era basar el treball en un dels mètodes de Zhang et al. [6], per obtenir els tràfics es va intentar des de cercar a bases de dades d'Internet en formats PCAP sense èxit, ja que s'havia d'assolir el segon problema, la quantitat de tràfic; fins a implementar un simulador que posteriorment es va descartar per raons que s'exposaran al punt 4.1.1. Raons paregudes varen servir per descartar la utilització el protocol de simulació CCEAP [50], tal com es veurà posteriorment. Una altra possibilitat, també descartada, era comprar l'equipament necessari per tenir el mateix tràfic que plantejaven els investigadors però exigia unes despeses fora de l'abast del projecte. La solució als dos primers problemes va resultar, donat l'estat de l'art aconseguit durant el desenvolupament inicial del projecte, dissenyar i implementar un mètode no descrit a la literatura d'esteganografia en xarxa basat en el mètode de Zhang et al. [6] i que complís la mateixa premissa de indetectabilitat que proposen a la publicació esmentada.

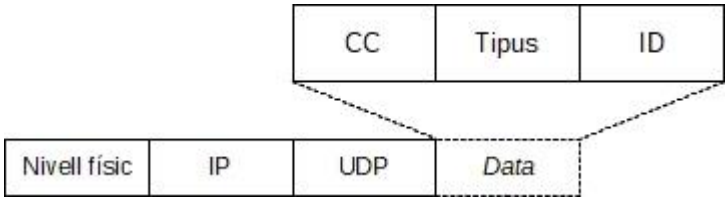
El darrer problema, la implementació del classificador, en aquest capítol es descriuran els aspectes bàsics de l'eina però la part d'estegoanàlisi s'explicarà amb detall al capítol 5.

4.1 Mètode per ocultar un missatge

Per si cal recordar què vol dir l'alteració de l'ordre, és una tècnica que consisteix a reordenar les PDU de forma diferent a com transitaria el tràfic segons surt de l'aplicació. En aquest punt s'exposen les tres formes que es van estudiar, dues de les quals es van descartar.

4.1.1 Simulador

La idea inicial era crear un simulador basat, tal com s'havia presentat a la primera entrega, en un dels dos mètodes de la publicació de Zhang et al. [6]. Per això, s'havia dissenyat una PDU de tres camps:

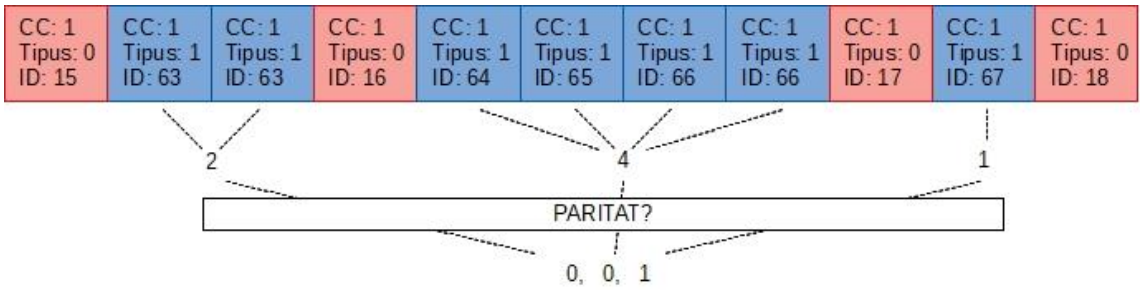


Imatge 4.01. PDU del simulador.

- CC: {0, 1}. 1 byte. Indicava si portava missatge ocult (1) o no (0).
- Tipus: {0, 1}. 1 byte. El mètode es basava en dos PDU, per tant, es simulava que una era de vídeo (1) i una altra d'àudio (0).
- ID. 4 bytes. Era un valor numèric real incremental diferent per a cada tipus.

El programari creava aquestes PDU cada vegada que s'havia d'enviar i les inseria en segments UDP que permetia el trànsit cap al destí. Com a simulador que era, no es requeria res més que els camps de control, ja que el que es volia estudiar era l'ordre, no el contingut.

Hi havia quatre opcions d'execució: com emissor o com receptor, i dins aquestes, amb un missatge ocult o sense. Pel cas d'enviar un missatge ocult, l'emissor amagava la informació considerant la paritat entre ID de les PDU que simulaven esser d'àudio quan hi havia una PDU de vídeo que les separava.



Imatge 4.02. Conversió del simulador.

Es va generar molt de tràfic i una vegada aconseguit, es va implementar la part d'enviament sense missatge ocult segons la distribució del tràfic amb missatge ocult. És a dir, es va obtenir una mitja de cada diferència entre ID i es va convertir en pesos per a crear un tràfic aleatori. Aquest pes, a més a més, es podia simular més o menys semblant al tràfic amb missatge ocult en funció d'un percentatge passat com paràmetre. D'aquesta forma hi havia molt de tràfic que complia:

- Si era amb missatge ocult, l'algorisme descrit per Zhang et al. [6].
- Si no ho era, el tràfic generat tenia una distribució o un comportament tan semblant a l'altre com es desitgés.

La idea era poder estudiar el problema de la detecció però a la inversa, ja que l'algorisme d'ocultació estava donat i era clar i simple. És a dir, el referent no era el tràfic sense missatge ocult si no el tràfic que servia com a portador d'un *network covert channel*.

El simulador finalment es va descartar perquè no es podia garantir que la distribució del tràfic que es volgués simular fos realment la que s'obtenia amb aquest programari especialment per la part del tràfic que no porta cap missatge ocult i, per tant, es podria arribar a conclusions que no es donarien al món real si es pogués accedir al tràfic real.

4.1.2 Protocol CCEAP

És una eina desenvolupada pel Dr. Steffen Wendzel i pel Dr. Wojciech Mazurczyk per estudiar diferents mètodes de *network covert channel* mitjançant un protocol creat per a tal efecte. Així, s'implementa una capçalera que porta tots els camps necessaris per generar tràfic amb missatges ocults segons els següents mètodes:

- *Storage*
    - Modulació de la grandària (vegeu 2.2.1.1)
    - Modulació de la seqüència (vegeu 2.2.1.2)

- Afegir redundància (vegeu 2.2.1.3)
- Valors aleatoris (vegeu 2.2.1.4)
- Modulació de valors (vegeu 2.2.1.5)
- Ús de bits reservats o inutilitzats (vegeu 2.2.1.6)

- *Timing*
  - Taxa o rendiment del tràfic de xarxa (vegeu 2.2.2.1)
  - Temps entre PDU (vegeu 2.2.2.2)
  - Pèrdues artificials (vegeu 2.2.2.4)
  - Retransmissions artificials (vegeu 2.2.2.5)
  - Manipulació de l'ordre (vegeu 2.2.2.6)

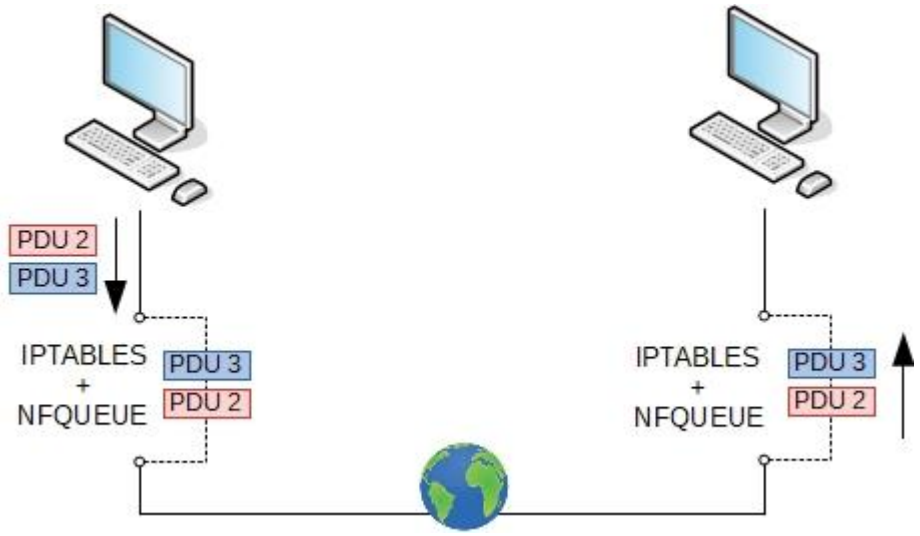
Es pot veure que podria ser d'interès per aquest projecte en el sentit que podia servir com un simulador generant el tràfic necessari per analitzar l'ordre de les PDU, ja que implementen aquest tipus de *covert timing channel*. Així, s'hauria haver d'estudiar bé aquest protocol i el seu funcionament per intentar veure si hi havia possibilitat de traslladar els resultats de forma fiable a l'objectiu del treball.

En aquest punt varen sorgir els mateixos dubtes que pel cas explicat a l'apartat 4.1.2 atès que el més important és no quedar-se al món teòric sinó donar la passa d'implementar una cosa funcional i vàlida en un entorn real. Per tant, igual que el simulador, es va descartar com font del conjunt de dades necessari per assolir els objectius del projecte.

4.1.3 Generació del *network covert channel*.

Cal recordar que l'objectiu del projecte és detectar *network covert channels* que utilitzen la tècnica de l'alteració de l'ordre de les PDU. Per obtenir un conjunt de dades amb el que treballar, es va decidir finalment dissenyar un mètode que permeti ocultar un missatge dins un flux de comunicació, que és el que es presenta en aquest apartat.

El mètode que es proposa per enviar missatges ocults utilitza el protocol *Real-time Transport Protocol* (RTP) implementat pel programari *Linphone* mitjançant la captura amb IPTABLES i NFQUEUE dels paquets de forma que s'altera l'ordre segons un algorisme conegut per l'emissor i el receptor i que es basa en el nombre de PDU enviades dels dos tipus (vídeo i àudio).



Imatge 4.03 Esquema general.

4.1.3.1 Descripció

El programari té una sèrie de requeriments que es detallen a continuació:

Requisit	Versió
Sistema operatiu	*GNU/Linux. Provat amb Linux Mint 19.3 i Ubuntu 16.04 LTS*
Intèrpret	*Python 3.6*
NFQUEUE	*fnfqueue-1.1.1*
IPTABLES	*1.6.0*
Linphone	*3.6.1*

Taula 4.01. Requisits.

a) RTP

Serveix per transmetre àudio i vídeo per Internet. Des del punt de vista del projecte, les característiques més importants són:

1. Utilitza UTP com protocol de transport.
2. Permet detectar pèrdues i errors d'ordre de PDU i compensar retards.
3. L'especificació RFC 3550 especifica dos protocols, RTP i RTCP, però només cal centrar-se en RTP per aquest projecte.

4. Les dades d'àudio i vídeo se separen segons el camp *Payload Type* (PT) de la capçalera (7 bits).

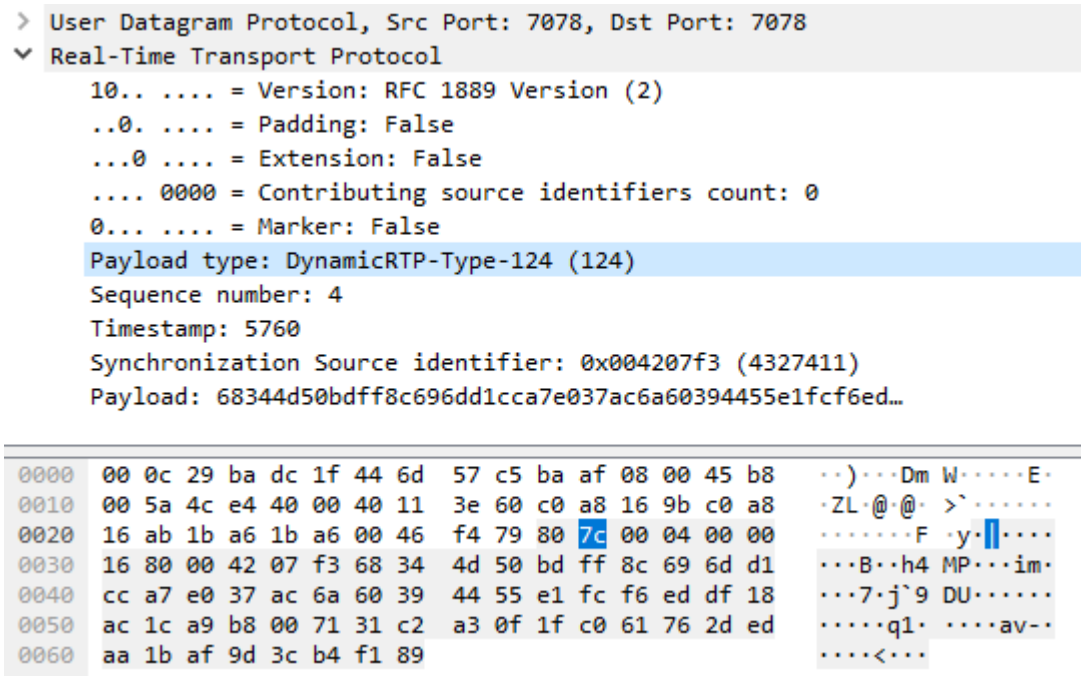
5. És un protocol punt-a-punt a nivell IP i, per tant, els paquets s'envien entre els destins finals.

b) *Linphone* (v.3.6.1)

És una aplicació que compleix el protocol de forma pura i permet tant trucades amb comptes registrats per Internet de forma gratuïta com en entorns locals sense comptes [51]. D'aquesta forma es poden fer proves tant en local com en entorns reals, passant tallafocs i altres elements de seguretat.

La versió instal·lada utilitza els còdecs *opus* per àudio i *VP8* per vídeo de forma que el tipus queda reflectit com 124 i 103 respectivament. Cal destacar que utilitza els ports UDP 7078 i 9078 per a enviar cada tipus de PDU però aquests ports també s'usen per a altres tipus de tràfic i per això no serveix la categorització per port si no que s'ha de comprovar el tipus al camp PT de la PDU RTP. No obstant això, conèixer els ports serveix per capturar el tràfic.

També és important veure que el tràfic en aquesta aplicació no s'envia amb la seqüència d'una PDU de vídeo seguida de N d'àudio sinó que envien M de vídeo i N d'àudio, on N i M són nombres sencers majors que zero.



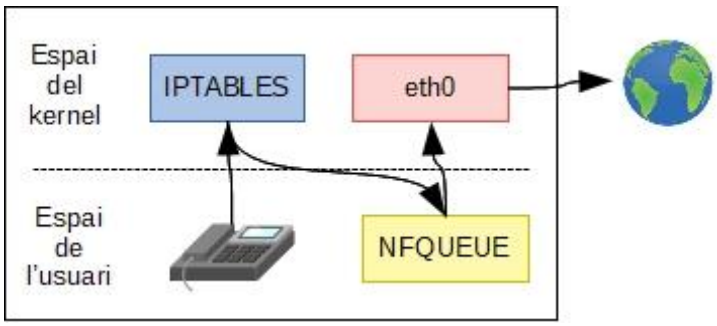
Imatge 4.04. Captura PDU d'àudio amb el programa *Wireshark*.

c) IPTABLES

És un programari que permet a un administrador de qualsevol sistema operatiu que es secundi sobre el nucli o *kernel* de Linux configurar regles per filtrar paquets IP [52]. A efectes pràctics es pot considerar un tallafoc a escala local perquè és el seu ús més estès però la seva organització, en taules que gestionen diferents regles, permet altres usos. Un d'ells s'aconsegueix mitjançant la taula mangle que permet capturar un paquet i alterar-lo abans d'enviar-lo al nucli del sistema operatiu per al seu processament cap a la xarxa física.

d) NFQUEUE

És un mòdul del *kernel* de Linux que permet gestionar els paquets a l'espai de l'usuari. A baix nivell utilitza *sockets* per la comunicació bidireccional i és una solució àmpliament usada per a un ISP, com aleshores, *suricata* [53]. Té les següents dependències: *python3-dev* i *fnfqueue* (instal·lat amb *pip3*).

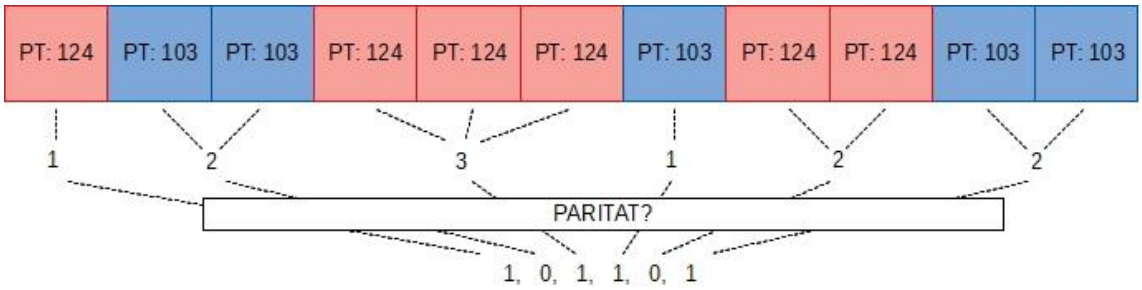


Imatge 4.05. Esquema IPTABLES i NFQUEUE.

e) Algorisme

Una vegada es poden capturar els paquets IP de la conversa que resulta rellevant per al projecte, l'emissor ha d'alterar-los de forma que el receptor pugui entendre el missatge ocult.

L'algorisme es basa a codificar la paritat del nombre de PDU enviades seguides amb el mateix tipus de contingut multimèdia (això és, àudio o vídeo). És a dir, un zero es codifica amb un nombre de PDU enviades parell i un ú amb nombre imparell. Cal destacar que el receptor no ha de fer res més que contar el nombre de PDU del mateix tipus seguides i obtenir la paritat. Aquest valor es va afegint a una variable que després es converteix per obtenir el missatge original (atès que no hi ha encriptació).



Imatge 4.06. Codificació de zeros i us.

El pseudo-codi de l'algorisme que es proposa quan es vol enviar un missatge ocult és el següent:

```
Per a cada paquet IP:
      s'obté el tipus de PDU
      si el tipus actual és igual que el de l'anterior i tocava enviar-ho
            comptador + 1
            s'envia l'anterior
      si no són iguals i cal enviar del tipus anterior
            es calcula la paritat amb la variable comptador
            si hi ha paritat
                  comptador = 0
                  s'envia l'anterior
                  toca enviar el tipus de l'actual
            si no n'hi ha però el comptador = 0
                  comptador = 1
                  s'envia l'anterior
                  s'emmagatzema paquet actual com anterior pel següent torn
            si no
                  s'emmagatzema l'anterior a la cua del seu tipus
                  s'emmagatzema paquet actual com anterior pel següent torn
                  comptador = 0
                  toca enviar el tipus actual

            si del tipus que cal enviar hi ha paquets a la cua
                  comptador = 0
                  mentre la cua no estigui buida
                        s'envia el paquet
                        comptador + 1
      si no
            s'emmagatzema l'anterior a la cua del seu tipus
```

Per tant, es basa a comparar les PDU que arriben amb les que estan retingudes i, en funció del tipus i de la paritat, s'envien unes o les altres però la que acaba d'arribar sempre s'emmagatzema a la memòria i s'envia alguna que estava emmagatzemada. D'aquesta forma l'algorisme no és susceptible a les compensacions per retards, pèrdues o alteracions de l'ordre, ja que la implementació de RTP ho fa sobre el nombre de seqüència.

Cal tenir present que l'objectiu del projecte no és dissenyar un bon mètode d'esteganografia en xarxa ja que aquest no és realment fiable atès que es fa sobre UTP i requeriria la implementació d'una capa de control o la modificació de l'algorisme donant prevalença al nombre de seqüència per assegurar l'arribada dels paquets IP.

4.1.3.2 Proves realitzades

Les proves per verificar la funcionalitat de l'algorisme tant en l'àmbit de l'emissor com del receptor es varen fer a una xarxa local amb sistemes operatius *GNU/Linux*, *iptables* v1.6.1 i python3.6 per a executar el codi. La versió de *Linphone* és molt rellevant perquè a la darrera versió en fase *beta* (v4.2) per defecte tot el tràfic es mou amb el mateix tipus de PDU.

Abans d'iniciar una conversa, cal interceptar el tràfic amb IPTABLES i NFQUEUE. Amb permisos de superusuari, al costat de l'emissor cal executar:

```
$ sudo iptables -t mangle -A POSTROUTING -p UDP --match multiport    --dports
7078,9078 -j NFQUEUE --queue-num 1
```

Al costat del receptor:

```
$ sudo iptables -t mangle -A INPUT -p UDP --match multiport --dports
7078,9078 -j NFQUEUE --queue-num 1
```

El programa desenvolupat s'anomena *tfm_cc_rtp.py*, també requereix permisos de superusuari per accedir a la cua creada per NFQUEUE i el codi font està disponible a l'annex B.

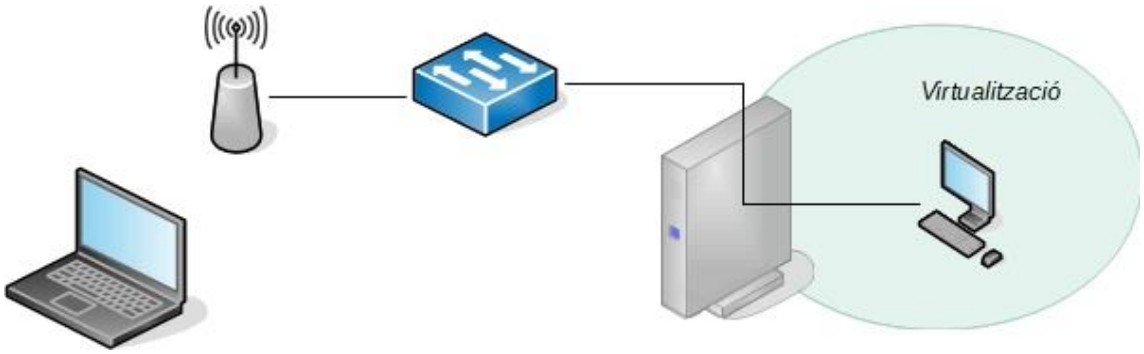
```
$ sudo python3 tfm_cc_rtp.py [-v|-d] (decoder|encoder (ncc|cc
['secret_message'|-t file_name]))
```

Paràmetres / Opcions	Primer	Segon	Tercer	Quart
Enviar missatge ocult	*encoder*	*Cc*	*'missatge'*	-
			-t	*Nom_del_fitxer*
Enviar sense alteracions	*encoder*	*Ccc*	-	-
Rebre missatge ocult	*decoder*	*Cc*	-	-
Rebre sense alteracions	*decoder*	*Ncc*	-	-

Taula 4.02. Opcions del programa que oculta els missatges.

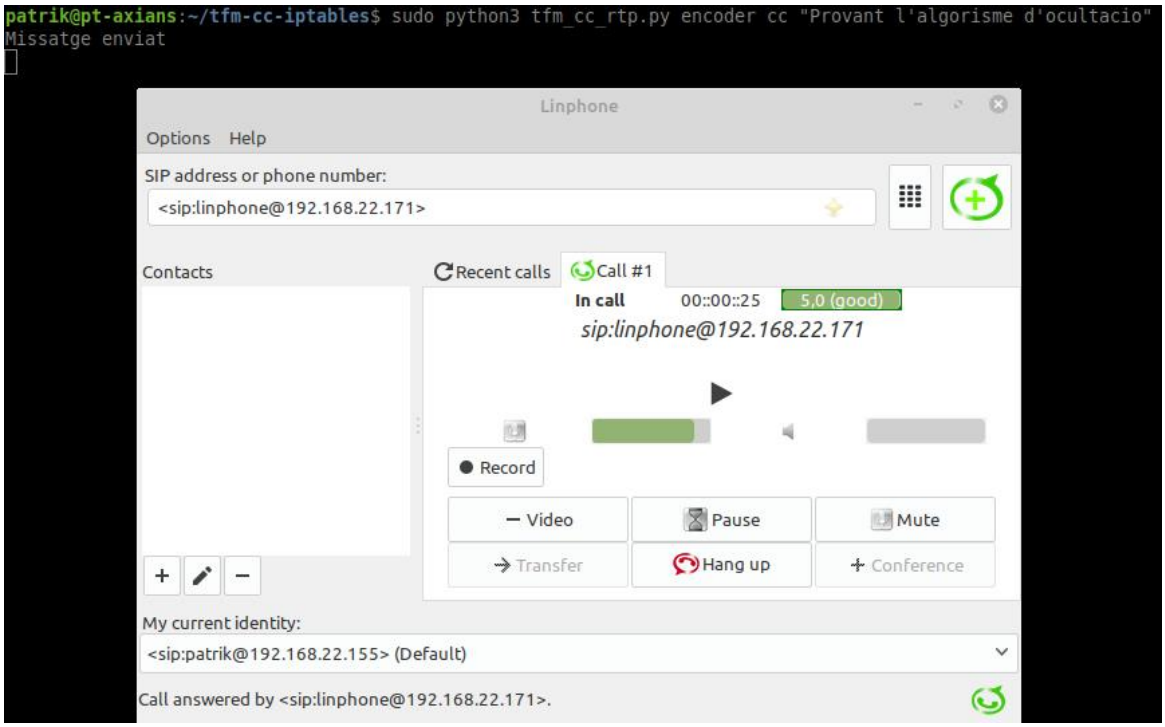
A la taula 4.02 no s'ha reflectit que hi ha dos paràmetres opcionals que s'han de col·locar els primers que serveix per indicar el grau de missatges que ha de treure el programa. Bàsicament el que fa és mostrar el nombre de PDU enviades i variables que serveixen per prendre decisions.

Pel que fa a la infraestructura de proves, es van fer amb un portàtil connectat per *WiFI* i una màquina virtual dins un altre ordinador tal com es pot veure a la figura 4.07.

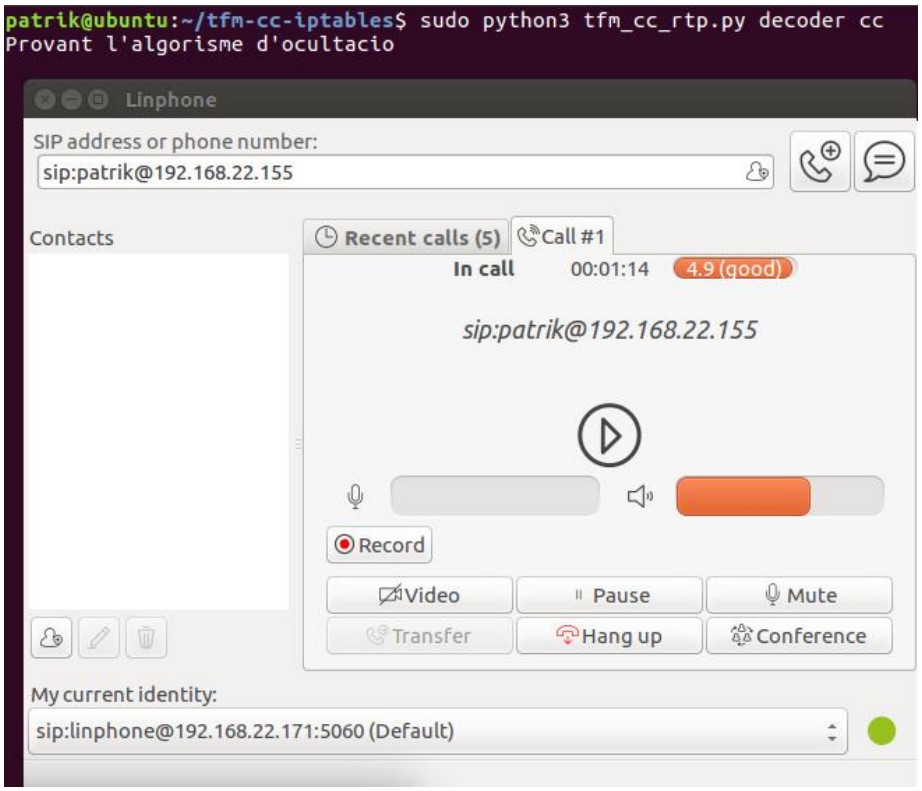


Imatge 4.07. Esquema de connexions a la xarxa local.

El resultat de l'enviament i de la recepció es poden veure a les següents imatges.



Imatge 4.08. Prova enviament a una xarxa local.



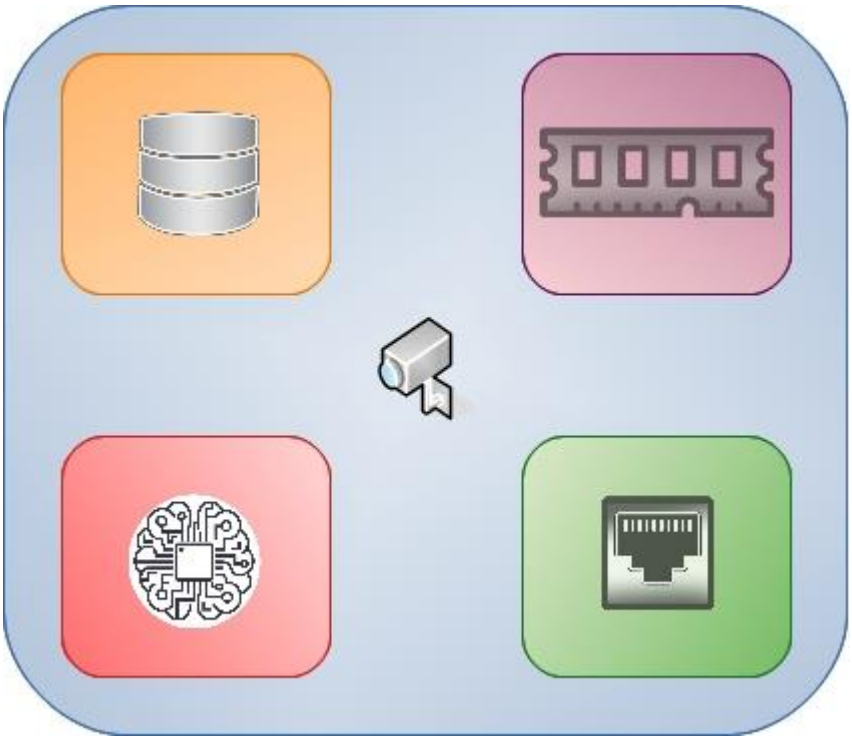
Imatge 4.09. Prova de recepció a una xarxa local.

Com ja s'ha explicat, ni el programari ni el mètode per ocular la informació ofereixen robustesa, és a dir, no garanteixen que arribi el missatge segons s'ha enviat. No obstant això, a l'hora de fer recollir les dades per entrenar el detector es va garantir que les PDU enviades eren totes capturades situant un *sniffer* al mateix dispositiu que originava el tràfic alterat.

4.2 Detector implementat

L'objectiu d'aquest programa és gestionar tràfic de xarxa per a comprovar si té missatges ocults mitjançant *network covert channels*. Per aconseguir-ho, es divideix en mòduls els quals tenen funcions diferenciades, corren de forma paral·lela amb una comunicació interna i se secunden en programes de tercers.

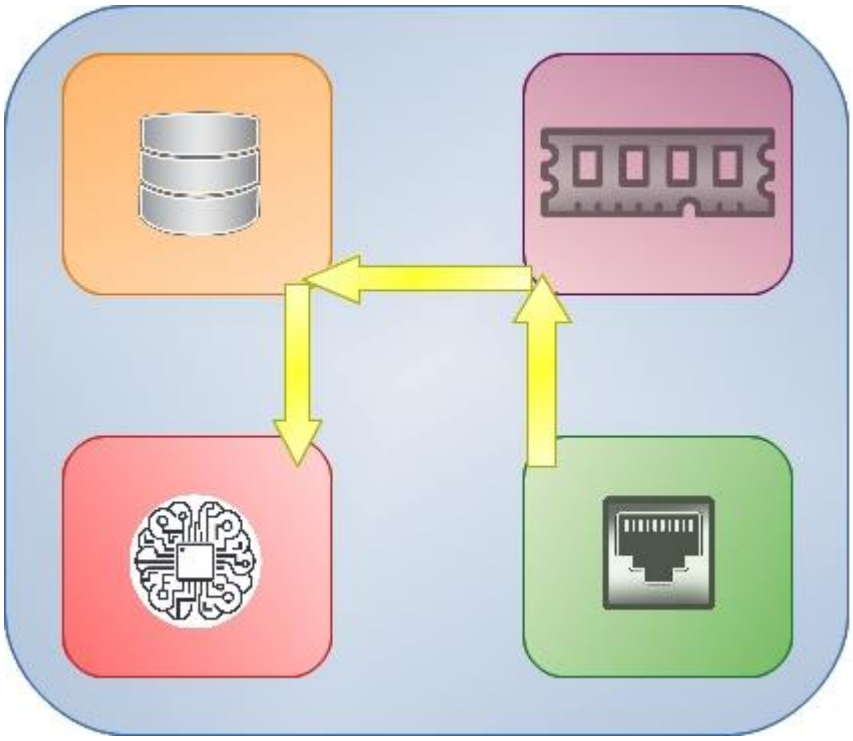
El disseny modular permet tant fer actualitzacions del codi sense tocar la resta del programari com focalitzar recursos a les tasques que més ho requereixen. Així, els mòduls amb tasques més crítiques o sensibles, s'executen en processos paral·lels i deriven funcionalitats en altres que no afectaran la pèrdua d'informació en cas de congestió en l'equip amfitrió.



Imatge 4.10. Disseny modular.

Una altra de les característiques interessants és que permet tres formes d'execució: com *daemon* escoltant una interfície de forma contínua fins que un administrador del sistema decideix aturar el programa, llegint un fitxer PCAP que incorporarà a la base de dades per ser tractat pels mòduls d'anàlisi de l'aplicació, o per entrenar l'algorisme amb la base de dades de tràfic. D'aquesta forma, es pot tenir el programari analitzant el tràfic en temps real i a la vegada generar un nou classificador cada cert temps amb el tràfic capturat o revisar un tràfic concret que hi ha emmagatzemat a un fitxer PCAP.

El flux del programari depèn del mode d'execució. A la figura 4.11 es veu què passa amb un paquet que arriba a la interfície en temps real. Primer es passa a una cua específica del protocol que sigui. Un procés d'aquest protocol llegeix la cua i introdueix el paquet a la base de dades. Un altre procés revisa la base de dades creant identificadors per converses, de forma que l'extracció posterior del flux de dades sigui segons aquesta anàlisi previ. Finalment, cada flux és analitzat per determinar si s'està utilitzant com un *network covert channel*.



Imatge 4.11. Recorregut d'un paquet.

Cal destacar que l'eina implementa les funcionalitats bàsiques els protocols TCP, UDP i IP, a més a més d'analitzar tot el tràfic RTP.

4.2.1 Requisits

Abans d'avançar, cal explicar els requisits principals del programari:

Requisit	Versió
Sistema operatiu	*Linux Mint 19.3*
Intèrpret	*Python 3.6*
Base de dades	*MongoDB 3.6.3*
Llibreries	*pymongo 3.9.0*
	dpkt 1.9.2
	numpy 1.17
	scikit-learn 0.23.0

	multiprocessing

Taula 4.03. Requisits.

- *MongoDB* (v3.6.3)

  És un sistema de base de dades *NoSQL* i, en concret, orientat a documents. Les seves característiques permeten inserir els paquets IP sense tenir present el format concret ni el volum de dades que contenen [54].

- *pymongo* (v3.9.0)

  Llibreria de *Python* per comunicar-se amb la base de dades *MongoDB* [55].

- *dpkt* (v1.9.2)

  És un mòdul de *Python* utilitzat en aquest projecte per interaccionar amb fitxers PCAP. Emmagatzema cada nou paquet com un conjunt independent de bytes amb el qual es pot treballar fàcilment [56].

- *Numpy* (v1.17)

  És una llibreria per *Python* que permet tractar grans volums de dades en forma de matrius multidimensionals. Implementa nombroses funcions sobre aquestes dades, entre les quals hi ha funcions estadístiques [57].

- *Scikit-learn* (v0.23.0)

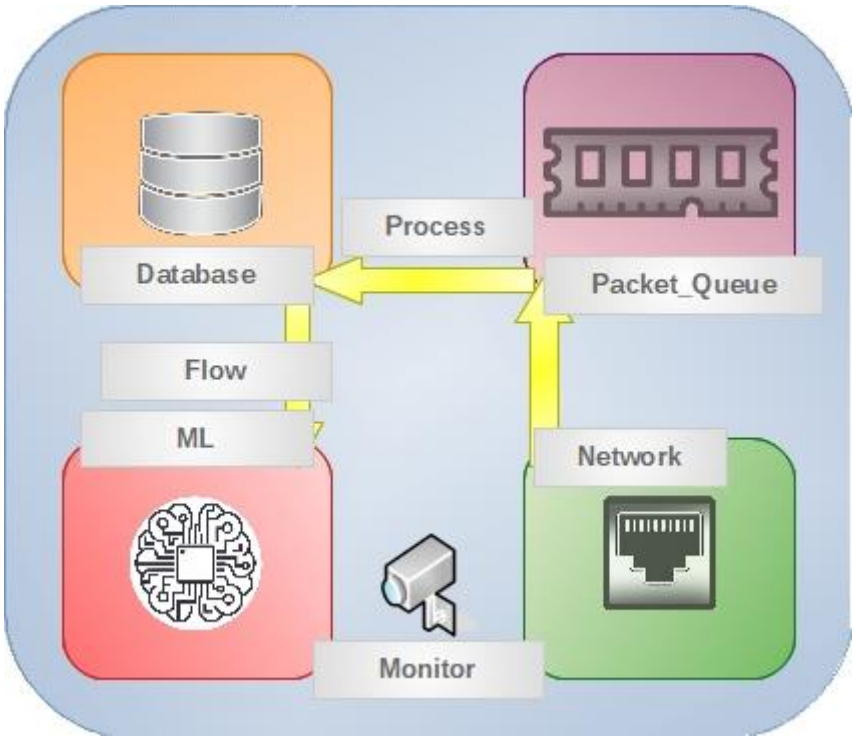
  És una llibreria per *Python* que implementa diversos algorismes de *machine learning* entre els quals estan tots els que s'han anomenat al capítol 3. Està dissenyat per treballar amb dades de la llibreria *Numpy* [58].

- *multiprocessing*

  És una llibreria de *Python* que permet executar processos de forma paral·lela. Implementacues que permeten la comunicació entre processos [59].

4.2.2 Mòduls i funcionalitat

Els mòduls o classes implementats són:



Imatge 4.12. Relació entre els mòduls principals.

- ***ML***

  És el mòdul que entrena i classifica els fluxos de tràfic. El seu funcionament es detallarà al capítol 5.

- ***Database***

  Permet la interacció amb la base de dades. Les seves funcions estan separades per protocol per evitar decisions sobre variables que es poden controlar fàcilment i reduir així els recursos necessaris.

- ***Flow***

  Gestiona el tràfic per protocols creant un identificador únic per a cada conversa. Segons el protocol, la conversa es gestiona de formes diferents. Per exemple, per a les PDU RTP es consideren les IP d'origen i destí, ja que si se separen per ports, no es podria analitzar la relació entre els dos tipus de PDU. L'evolució futura d'aquest mòdul és una màquina d'estats que permeti eliminar els identificadors i el tràfic de la base de dades quan es consideri que no porta cap missatge ocult.

- ***Init***

  Inicia els diferents processos i gestiona la comunicació entre ells. Com s'ha explicat, el nombre de processos depèn de la responsabilitat de

les tasques assignades. Per exemple, per processar els segments TCP es creen deu processos per defecte, per UDP són cinc i per ICMP un.

- ***Monitor***

  S'encarrega de monitorar els recursos del sistema i l'estat dels fluxos i la base de dades. L'evolució d'aquest mòdul és que pugui prendre decisions sobre el nombre de processos necessaris en funció dels recursos utilitzats al sistema operatiu. Se secunda en les dades del mòdul *Monitor_Data.*

- ***Monitor_Data***

  És un mòdul que serveix com a base de dades en memòria de les variables de control, com l'estat de les converses derivades del tràfic de xarxa.

- ***Network***

  Escolta la interfície configurada i passa els paquets a la cua del seu protocol.

- ***Packets_Queue***

  Implementa l'emmagatzematge en memòria dels paquets segons arriben a la interfície. No es fa cap tractament.

- ***Pcap***

  Llegeix els paquets d'un fitxer PCAP i els passa a la cua de missatges per protocol.

- ***Process***

  Està pendent de l'arribada de nous paquets a les cues per passar-los a la base de dades amb el format adient perquè posteriorment puguin ser tractats. Es creen processos pels principals protocols del model TCP/IP.

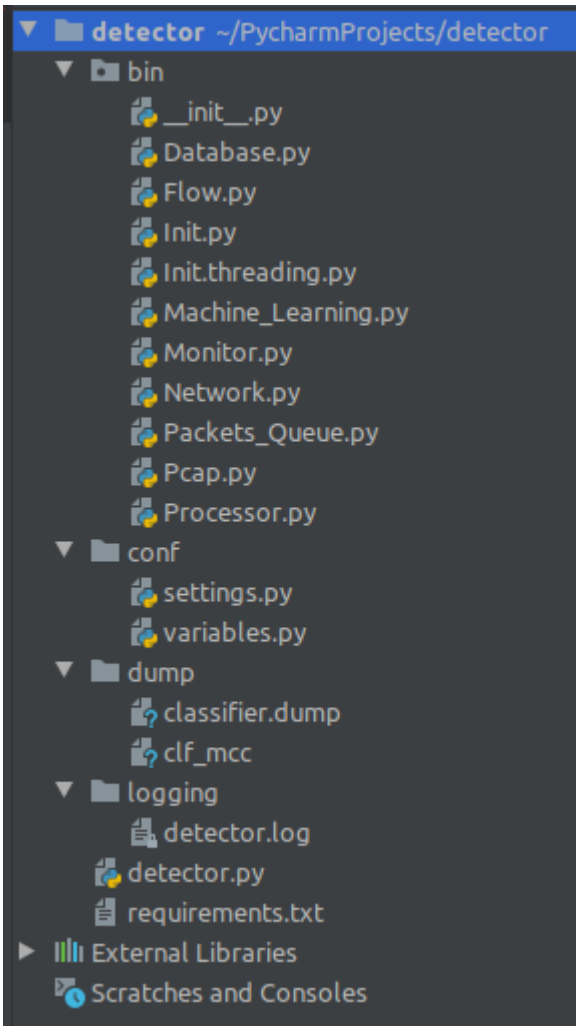
Com ja s'ha descrit de forma esquemàtica, el programa està pensat per llegir la interfície de xarxa i no perdre temps amb el processament de cada paquet. Així, una vegada llegit (*Network*), passa a la memòria RAM *(Packets_Queue*) d'on un altre procés (*Process*) l'extrau per bolcar-lo a la base de dades (*Database*) que està al disc dur.

Una vegada està al disc dur, es processa en conjunt amb la resta de paquets de la corresponent conversa. Primer, es crea un identificador únic i un estat inicial per a cada conversa (*Flow*) i a partir d'aquest punt és el procés de *machine learning* (*ML*) el que s'encarrega de gestionar el tràfic.

L'algorisme de *machine learning* farà prediccions en funció d'un entrenament i mostra una alerta quan es considera que una conversa pot portar un missatge ocult. Tot això es veurà en detall al següent capítol.

4.2.3 Estructura

En aquest apartat es veuen els fitxers i directoris del programa:



Imatge 4.13. Estructura.

- *detector.py*: executable inicial.
- *requeriments.txt*: fitxer amb les llibreries que cal instal·lar.
- *logging*: directori que conté un únic fitxer amb els missatges que el programari considera que s'han de registrar.
- *bin*: directori dins el qual hi ha el codi dels mòduls explicats.

- *conf*: directori que conté els fitxers de configuració, que són:
    - *settings.py*: variables d'execució del programa com les dades de connexió a la base de dades, la interfície que cal escoltar o els paràmetres PPD.
    - *variables.py*: especifica el valor de les constants utilitzades durant l'execució.
- *dump*: emmagatzema els fitxers amb el classificador i el valor de la mètrica MCC associada.

4.2.4 Modes d'execució

Els modes d'execució s'implementen amb els paràmetres obligatoris que s'han de passar al programa a l'hora d'executar-lo. Les tres possibilitats impliquen diferents funcionalitats que tenen distints fluxos d'execució. Encara que no s'expliquen quant al codi, és important conèixer les directrius principals de cadascun.

Paràmetres Opcions	Primer	Segon	Tercer
Processa el tràfic un fitxer PCAP	*-p / --pcapfile*	*Fitxer*	[*covert* | *overt*]
Entrena l'algorisme de *machine learning*	*-t / --training*	-	-
Analitza el tràfic en temps real	*-d / --daemon*	-	-

Taula 4.04. Opcions del programari *detector*.

Les particularitats de cada opció són:

- *pcapfile*: llegeix el fitxer PCAP, fica el tràfic a la base de dades i pot analitzar-ho. No requereix permisos especials d'execució. Té dos paràmetres dels quals només un és obligatori:
    - *Fitxer*: obligatori. Nom del fitxer que conté el tràfic que es vol analitzar. S'ha de posar la ruta sencera de l'estructura de fitxers del sistema operatiu.
    - *covert* | *overt*: opcional. Permet especificar si el tràfic capturat té un *missatge ocult* o no. Si s'especifica, es considera que és tràfic per entrenar l'algorisme i es guarda a la base de dades

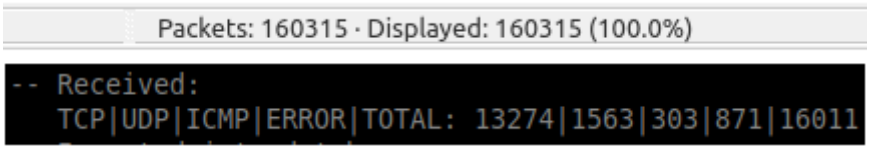
per poder fer l'entrenament. Si no, es realitza l'anàlisi del tràfic des del punt de vista de l'esteganografia en xarxa emmagatzemant-ho a una base de dades alternativa que posteriorment s'elimina.

- *training*: permet entrenar l'algorisme de *machine learning* i volcar el resultat, si és millor que l'anterior, al fitxer corresponent. Utilitza la base de dades com font d'informació però només el tràfic que ja està categoritzat com *network covert channel* o no.

- *daemon*: analitza en temps real el tràfic que va capturant de la interfície de xarxa. El tràfic no es considera de primeres com *network covert channel*.

4.2.5 Proves realitzades

El primer que cal dir és que s'han fet proves des del punt de vista de la gestió de paquets i els resultats no han estat molt positius. Per exemple, en una comparació amb el popular *sniffer Wireshark* el resultat de captar uns 400Mbps ha estat que només es gestionaven el 10% dels paquets que circulaven per la xarxa amb el mateix ordinador al mateix moment.



Imatge 4.14. Comparativa.

Caldria una investigació amb l'altra llibreria de multiprocessament de *Python*, *threading* ja que té un perfil de més baix nivell [60] i tal vegada es podria separar l'execució de xarxa d'aquesta forma. La implementació es va fer amb *multiprocessing* perquè es va donar més importància a la correcta gestió dels recursos de processament del dispositiu físic atès que *Python* no està pensat per processadors multinuclis. No obstant això, per les proves realitzades per detectar tràfic a la xarxa local no es varen perdre PDU i els resultats es veuran al capítol 5.

Respecte de la base de dades és important explicar que no s'hi ha securitzat la connexió amb la base de dades per simplificar la configuració i entorn de proves. També cal destacar que atès que el volum d'informació que es maneja a una xarxa és molt elevat, a la consola de *MongoDB* s'ha de modificar el paràmetre que designa la quantitat de memòria disponible per realitzar ordenaments a la base de dades (per defecte, són 32MB).

```
> use admin
switched to db admin
> db.adminCommand({setParameter:1, internalQueryExecMaxBlockingSortBytes:
50151432})
{ "was" : 33554432, "ok" : 1 }
```

Per a comprovar-ho, només cal executar el següent comando:

```
> db.runCommand({getParameter:1, internalQueryExecMaxBlockingSortBytes:1})
{ "internalQueryExecMaxBlockingSortBytes" : 50151432, "ok" : 1 }
```

El màxim valor són 2GB (2147483647).

El codi font es pot veure a l'annex C i només requereix instal·lar les dependències amb *pip*:

```
 $ pip install –r requeriments.txt
```

Amb això, el programari es pot iniciar amb el següent comando:

```
$ python3 detector.py -h
```

```
patrik@CaNostra:~/PycharmProjects/detector$ python3 detector.py -h
usage: detector.py [-h] [--verbosity VERBOSITY]
                   (-d | -p [PCAPFILE [PCAPFILE ...]] | -t)

Network Steganography Detector based on OpenAI.

optional arguments:
  -h, --help            show this help message and exit
  --verbosity VERBOSITY
                        1 - ERROR, 2 - WARNING, 3 - INFO, 4 - INFO MACHINE
                        LEARNING, 10 - DEBUG
  -d, --daemon          start as a daemon over the configure interface in the
                        settings file
  -p [PCAPFILE [PCAPFILE ...]], --pcapfile [PCAPFILE [PCAPFILE ...]]
                        analyze a PCAP file specifying the traffic type
                        [filename [covert|overt]]
  -t, --training        train the ML algorithm
```

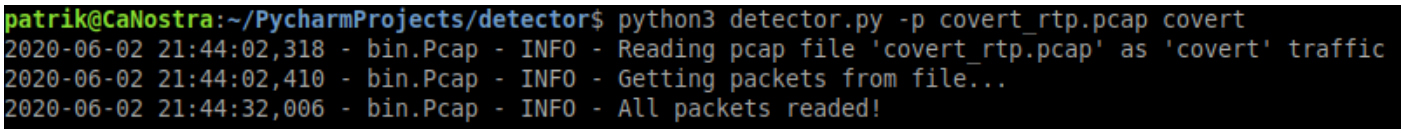
Imatge 4.15. Execució del *detector*.

Així, si es vol afegir un fitxer PCAP amb tràfic que porta un missatge ocult, el resultat serà el següent

```
patrik@CaNostra:~/PycharmProjects/detector$ python3 detector.py -t
2020-06-02 20:21:07,719 - bin.Machine_Learning - INFO - Training: getting data sets..
2020-06-02 20:24:51,658 - bin.Machine_Learning - INFO - Training: getting features..
2020-06-02 20:24:53,388 - bin.Machine_Learning - INFO - Init iteration number 0
2020-06-02 20:25:49,774 - bin.Machine_Learning - INFO - Classifier improved! New MCC metric value: 0.9051504417978317
```

Imatge 4.16. Afegir un fitxer amb nou tràfic.

Una vegada introduït, es pot fer l'entrenament per veure si es millora el classificador.



Imatge 4.17. Entrenar el classificador.

Al capítol 5 es mostrarà el funcionament general del programa des del punt de vista de la detecció, la qual cosa permetrà deduir el correcte funcionament d'aquesta eina per a totes les funcionalitats.

5. Estegoanàlisi

Aquest capítol final mostra els resultats d'aplicar els algorismes de machine learning al tràfic generat amb l'algorisme descrit al capítol 4 (vegeu 4.1.3) capturat i analitzat amb l'eina explicada al mateix capítol (vegeu 4.3).

Encara que els *network covert channels* són coneguts des dels anys setanta [3] i que la comunitat científica ha estudiat molt els protocols des d'aquest punt de vista des de l'any 2005, la indústria no ha tingut molt interès per implementar mètodes de detecció. És possible que influeixin els següents tres punts:

- Baixa utilització dels *network covert channels* per tres causes principals: es requereixen coneixements avançats, baixa taxa de transferència i molta susceptibilitat a problemes de transmissió.

- Ús de molts recursos pel que fa a la detecció: molts mètodes exigeixen consums de memòria i CPU elevats per tractar el tràfic de formes holístiques.

- Detecció individualitzada: el fet de no aconseguir la implementació un mètode únic per abastar totes tècniques d'ocultació.

No obstant això, està documentat que s'utilitzen activament en *programari maliciós* per ocultar tràfic C&C de les *botnets*, per coordinar atacs DDoS, per part de serveis d'intel·ligència o per filtrar informació classificada ([21] i [61]). Per tant, encara que el seu ús no sigui molt comú, les seves aplicacions són prou crítiques.

En aquest projecte s'ha intentat donar una passa més cap a un mètode de detecció de la tècnica que altera l'ordre de les PDU dins els *network covert timing channels* amb un mètode que ha donat molt bons resultats, tal com es veurà. El potencial que té aquest mètode recau de forma completa en la tècnica innovadora en esteganografia en xarxa per a l'obtenció de les *features*, anomenada *Patterns of Pixels Differences* (PPD) [62]. Aquesta tècnica s'explica amb detall a l'apartat 5.2

5.1 Anàlisi estadístic

A la introducció es va definir el concepte de indetectabilitat segons Cachin [8] que usa una mètrica basada en l'entropia entre dues funcions de distribució. El concepte és que dues funcions de probabilitat de tràfics de xarxa siguin molt semblants estadísticament i, d'aquesta forma, es pugui considerar que no hi haurà cap mètode basat en estadística que detecti les diferències.

Així ho plantegen Zhang et al. en [6] i en [63], els quals apliquen la investigació de Liu et al. [64] a la tècnica d'ocultació d'alteració de l'ordre del missatge. Així, el tràfic es pot considerar polinomialment indetectable si donades dues funcions de probabilitat de dos tràfics de xarxa diferents $T(n)$ i $T'(n)$ les diferències són insignificants respecte d'un paràmetre de seguretat σ.

$$|T(n) - T'(n)| \leq f(\sigma)$$

Com a part dels objectius del projecte, s'ha decidit comprovar si el mètode proposat al punt 4.1.3.1 compleix els requisits per ser considerat indetectable de la mateixa forma que ho fan Zhang et al. [6] atès que està basat en un dels mètodes proposats al mateix estudi i, d'aquesta forma,

poder valorar de forma més encertada els resultats del mètode de detecció.

Els dos tests presentats a l'apartat 3.1 són els que s'utilitzen amb la motivació esmentada.

- *KS test*: s'ha utilitzat la funció *ks_2samp* de la llibreria *scipy.stats* [65]. Segons la descripció oficial, es tornen dos valors:

  1) *statistic*: resultat del test, el qual mesura la distància que hi ha entre les dues distribucions.

  2) *p-value*: mesura la probabilitat que el resultat donat per *statistic* sigui possible atès que la hipòtesi nul·la sigui certa, és a dir, que les distribucions no tenen relació.

  Per tant, el resultat s'ha d'interpretar com que si el valor de la probabilitat *p-value* és elevat, el resultat del test a *statistic* és exacte per a distribucions amb menys de 10.000 mostres.

- *KLD test*: s'ha utilitzat la funció *kl_div* de la llibreria *scipy.special* [66]. La modificació és perquè la funció *log2* no existeix com tal en la llibreria *math* però es pot utilitzar *log(P, 2)* que és el mateix. El resultat és un array de les diferències per a cada interval i per aquest motiu es passa per la funció *mean* de la llibreria *numpy* de forma que es pugui analitzar la diferència mitjana entre tots els intervals.

Abans de passar a veure els anàlisis és important explicar que s'ha trobat interessant analitzar el mètode de tres formes. La primera forma consisteix a revisar les semblances entre cinc converses que pertanyen a la mateixa videotrucada i durant el mateix temps on una porta un missatge ocult i l'altre no. La segona manera és enfrontant les distribucions de converses diferents on una porta un missatge ocult i l'altre no. I la darrera compara distribucions que no porten cap missatge ocult però també de diferents converses. Al darrer apartat d'aquesta secció es presenten unes conclusions respecte als resultats obtinguts.

El codi font de les proves es pot veure a l'annex A, ja que no està integrat amb les eines desenvolupades. Aquest codi requereix que es posin els vectors que es volen analitzar a mà. En aquest cas, cal recordar que s'analitzen el nombre de PDU seguides dels mateixos tipus (àudio o vídeo) que s'envien. D'aquí endavant, aquest concepte es veurà reflectit sota el nom "grups de PDU".

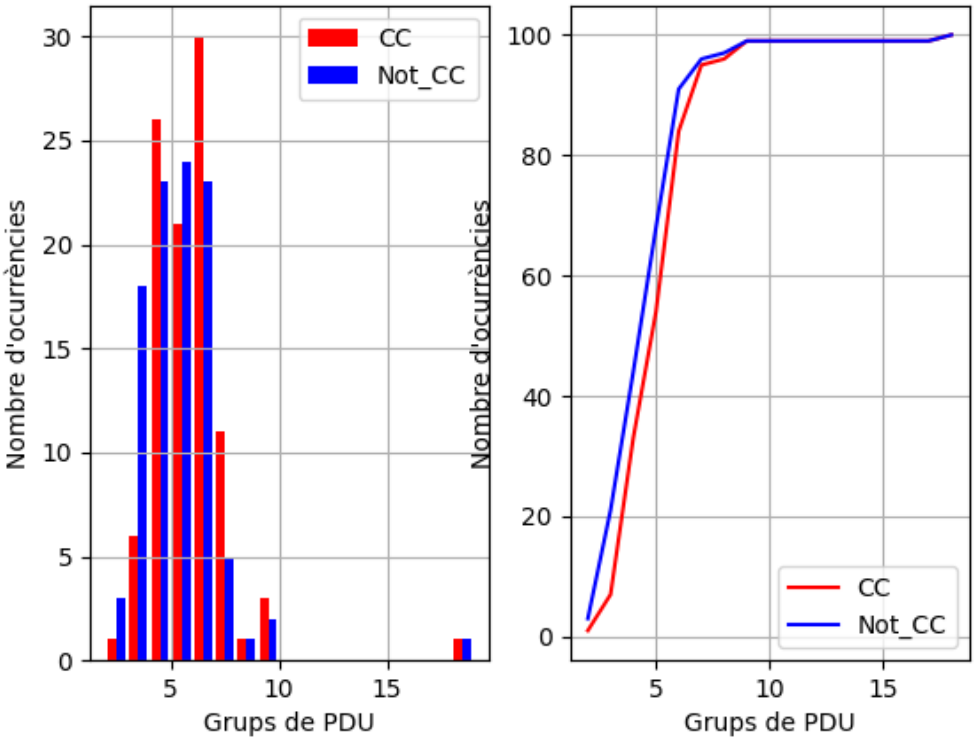
Pel que fa a l'ànalisi, tot es basa a enfrontar les distribucions del nombre de grups de PDU per a cinc conjunts de cent grups de PDU.

5.2.1 Mateixa videotrucada.

Tal com s'ha explicat, en aquest apartat es poden veure dues mostres de tràfic de 100 grups de PDU que són de la mateixa videotrucada al mateix moment per a cada distribució. Per aconseguir aquestes dades es va modificar el codi de l'algorisme del punt 4.1.3 per emmagatzemar a dos variables el nombre de PDU del mateix tipus seguides durant el transcurs de la videotrucada. A una d'elles es guardava el nombre de PDU segons arribaven a la cua (ja que així s'haguessin enviat a l'altre extrem) i a l'altre el nombre de PDU segons s'enviaven realment.

Aquesta informació permet veure quina és la diferència que introdueix l'algorisme presentat en aquest treball.

El gràfic de l'esquerra de la imatge 5.01 mostra la funció de distribució dels conjunts de dades testejats i la de la dreta és la seva distribució acumulativa. Als dos casos, l'eix horitzontal representa el nombre de PDU a cada grup del conjunt de dades. L'eix vertical varia en cada gràfica. A la de l'esquerra representa el nombre d'ocurrències de cada grup de PDU de forma individual. A la gràfica de la dreta, el nombre d'ocurrències acumulat. Es pot veure també que les dues gràfiques tenen una llegenda explicant els colors.



Imatge 5.01. Distribució d'una de les converses a la mateixa videotrucada.

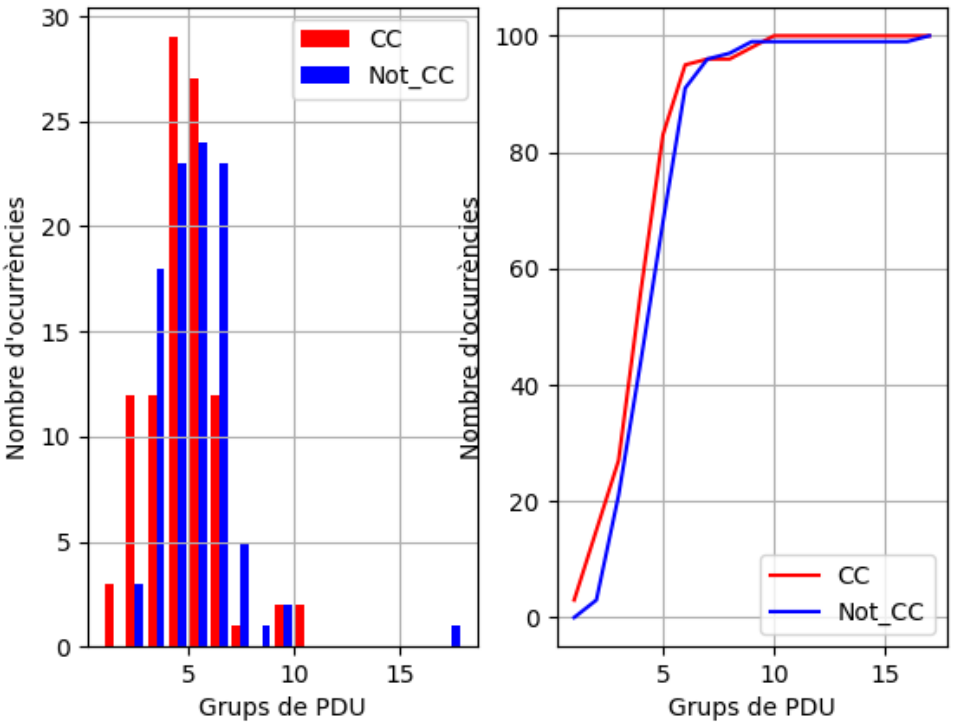
Converses	KS		KLD
	Resultat	p-value	
1	*0.1*	*1.0*	*0.7142244258723911*
2	*0.125*	*1.0*	*0.8168195389614885*
3	*0.2*	*0.9944575548290717*	*0.7554632271075462*
4	*0.1*	*1.0*	*0.8552791081450559*
5	*0.11764705882352941*	*0.9999438328545421*	*0.8232139076186944*

Taula 5.01. Resultats a la mateixa videotrucada.

Els resultats de la taula 5.01 mostren que pel que fa al test KS no es pot negar a cap conjunt de dades que pertanyen a la mateixa distribució donat un *p-value* tan a prop de la unitat i resultats al voltant del 0,1. Pel cas del test KLD els resultats encara que són bons, no ho són tant com els presentats a la publicació que serveix com referència de Zhang et al. [6].

5.2.2 Distintes videotrucades.

En aquest cas, els 100 grups surten de videotrucades totalment diferents. D'aquesta forma es pot analitzar les diferències entre dues converses tal com ho fa el detector implementat en aquest treball. El tipus de gràfiques és el mateix que a l'apartat anterior.



Imatge 5.02. Distribució d'una de les converses a diferents videotrucades.

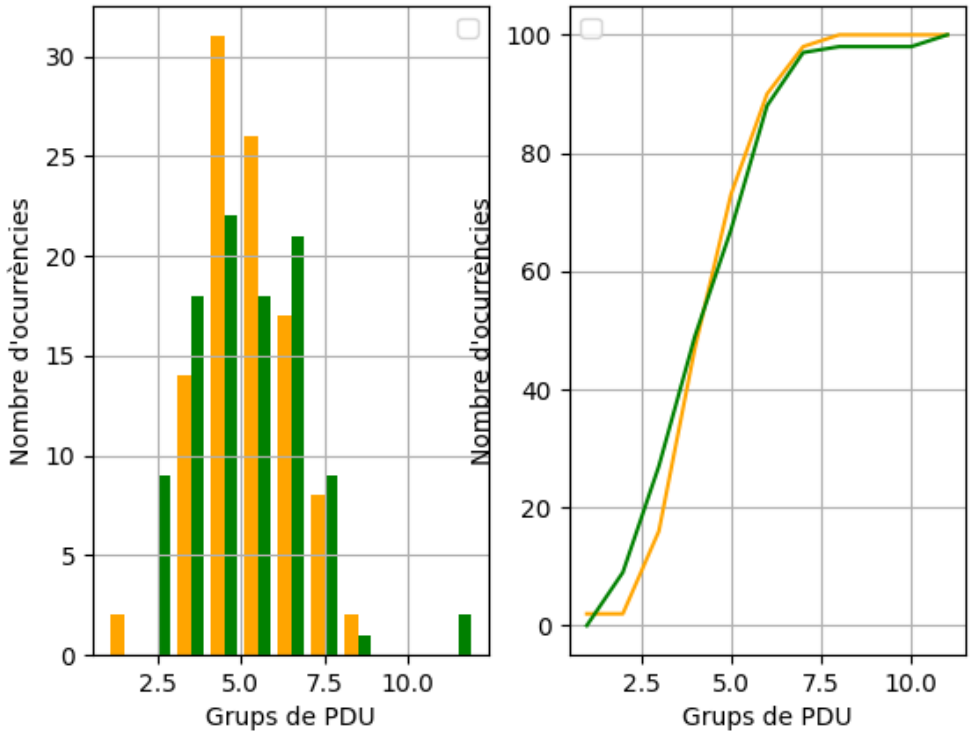
Converses	KS		KLD
	Resultat	p-value	
1	*0.11764705882352941*	*0.9999438328545421*	*0.9847809637635734*
2	*0.11764705882352941*	*0.9999438328545421*	*0.9926281769066293*
3	*0.11764705882352941*	*0.9999438328545421*	*0.815242689350332*
4	*0.11764705882352941*	*0.9999438328545421*	*1.0971211979571076*
5	*0.11764705882352941*	*0.9999438328545421*	*1.0498019629660689*

Taula 5.02. Resultats a diferents videotrucades.

En aquest cas els dos tràfics tenen resultats del test KS iguals perquè, cal recordar, mesura la màxima distància entre les distribucions però tots permeten afirmar que són de la mateixa distribució. Els resultats del test KLD està al voltant de 1,0, la qual cosa també és un resultat òptim.

5.2.3 Diferents videotrucades sense esteganografia

Ara es comparen dues distribucions de tràfic que no porta cap missatge ocult. Aquests resultats permeten analitzar les diferències intrínseques entre tràfics que pot haver-hi a les converses naturals d'aquest tràfic de xarxa. S'han alterat els colors perquè el tipus de tràfic no és el mateix que pels casos anteriors.



Imatge 5.03. Distribució sense missatge ocult a cap conjunt de dades.

Converses	KS		KLD
	Resultat	p-value	
1	*0. 18181818181818182*	*0. 9970968144342757*	*1.0952630669054901*
2	*0. 14285714285714285*	*0. 9999997041284721*	*0. 7862380537965584*
3	*0. 23076923076923078*	*0. 8978057035171046*	*1. 5407516602063855*
4	*0. 125*	*0. 9998909696588173*	*0. 06285749979286183*
5	*0.11764705882352941*	*0.9999438328545421*	*0. 6055907394890617*

Taula 5.03. Resultats sense canals encoberts.

Es pot veure que són resultats que confirmen també que les dues distribucions es poden considerar iguals, com de fet ho són.

5.2.4 Consideracions sobre l'execució

Cal tenir presents alguns aspectes sobre l'execució d'aquestes dues funcions respecte a les mostres agafades:

- S'ha vist que el primer grup de PDU de totes les videotrucades és d'àudio i solen ser unes 60 PDU. El segon grup és de vídeo i sol ser de 25 PDU. Si es fa els tests considerant aquests dos grups inicials, sempre surt que el tràfic no és igual (encara que l'algorisme no canviï els grups). Això és perquè hi ha molta desviació respecte de la resta de nombres de grups així que s'han obviat aquests dos grups inicials per fer les proves.

- Al test KLD no se li poden introduir mostres amb intervals que tinguin valor zero si no és zero també a l'altra distribució. La definició matemàtica presentada a l'apartat 3.1.2 permet deduir el motiu: hi ha una divisió. Per tant, el resultat es pot desviar a infinit o zero. Es pot solucionar amb una mostra de tràfic que sigui bona quant a qualitat tant de so com imatge i xarxa.

- Com més grans siguin els grups de PDU, els resultats mostren més diferències entre les funcions de distribucions. No obstant això, passa el mateix a totes les proves realitzades i presentades en aquest apartat.

5.2.5 Conclusions

En analitzar els resultats de les proves de forma conjunta, les conclusions respecte del concepte de indetectabilitat són:

- Com més petits siguin els resultats millor i els de la taula 5.01 representen que realment no s'introdueix una diferència significativa al flux de comunicació.

- Els resultats de la taula 5.03 mostra que hi ha conjunts de dades molt semblants però també n'hi ha molt diferents, ja que tot depèn de les necessitats derivades del flux RTP que s'està analitzant.

- Els resultats de la taula 5.02 entren dins els marges dels resultats de la taula 5.03.

D'aquesta forma es pot afirmar que el tràfic resultant no difereix estadísticament del tràfic original i, sota el paraigua de la literatura, es pot dir que és un mètode indetectable segons en Zhang et al. [6]. Era

important arribar a aquest punt per valorar el mètode de detecció de forma adient atès que s'han aconseguit resultats molt interessants.

5.2 PPD

5.1.1 Descripció.

Proposat per Lerch-Hostalot et al. [62] és una tècnica d'extracció de *features* per a estegoanàlisi en imatges basat en la relació entre els píxels que són veïns. Els investigadors estableixen un bloc de 3 x 3 píxels situant el punt central com una base per estudiar la diferència amb els píxels confrontats. D'aquesta forma es creen dues parelles horitzontals, dues verticals i quatre diagonals. Si cada parella pot prendre tots els valors dels colors, el resultat és una matriu excessivament gran per poder processar-la. És a dir, no és habitual que un píxel tingui el valor màxim (per exemple, 255) i el del voltant el mínim (0). D'aquesta forma, estableixen un llindar o valor límit $d(x, y)$ entre els dos píxels contigus *x* i *y*.

$$d(x,y) = \begin{cases} S-1, & si\ x < 0\ > S-1 \\ |x-y|, & si|x-y| \leq S-1 \end{cases}$$

On *S* representa el valor màxim de diferència entre els píxels. És a dir, els valors entre els píxels queden acotats entre 0 i $S-1$. No obstant això, les possibilitats segueixen sent molt elevades per a imatges grans així que van decidir reduir el nombre de parelles a quatre: una horitzontal, una vertical i dues diagonals. D'aquesta forma, aconsegueixen reduir les diferències de 256^9 a S^4 i així només cal estudiar les diferències entre el píxel central i quatre al seu voltant.

Aquestes diferències s'estudien mitjançant la creació de blocs que contenen els valors dels píxels. Si es defineix el bloc *B*:

$$B = \{110, 111, 110, 113, 112\}$$

Es poden estudiar les diferències de dues formes:

- Amb la diferència amb el valor mínim. És a dir, restant 110 a tots els valors. Al bloc anterior quedaria un vector amb els valors (0, 1, 0, 3, 2).

- Amb la diferència amb el valor màxim. És a dir, restant tots els valors a 113, la qual cosa donaria com vector resultant (3, 2, 3, 0, 1).

Ara bé, els investigadors van definir el valor màxim de diferència com $S = 3$ de forma que els vectors resultats són (0, 1, 0, 2, 2) i (2, 2, 2, 0, 1).

Si s'estudia tota la imatge generant diferències de la mateixa forma, el resultat serà un conjunt de vectors de quatre elements (eliminant el nombre del píxel central) els valors dels quals estaran entre 0 i $S - 1$. Les *features* sorgeixen d'aquests vectors.

5.1.2 Adaptació al tràfic de xarxa

Tal com es va veure al punt 4.1.3, s'oculta el missatge a la paritat del nombre de PDU seguides del mateix tipus. Per tant, les *features* que serviran per entrenar i decidir si un tràfic forma part d'un *network covert channel* han de sortir d'aquests agrupaments.

Adaptar de forma adient la tècnica al tràfic de xarxa és possible mitjançant les següents dues consideracions:

- La grandària del bloc *B*.

  Pel tràfic de xarxa descrit la relació es fa entre grups de PDU, és a dir, entre el nombre de PDU que van seguides d'un tipus i els següents. Si es considera cada grup com *x*, el bloc es pot definir com:

  $$B = (x_0, x_1, \dots, x_T), \qquad \forall\, T \in \mathbb{N} \mid T > 0$$

  On *T* representa el màxim nombre de grups que s'han de tenir en compte per a cada bloc. Així, per a $T = 4$ de la imatge 5.01 surten els següents tres blocs sencers:

  $$B_0 = (1, 2, 3, 1)$$
  $$B_1 = (2, 3, 1, 2)$$
  $$B_2 = (3, 1, 2, 2)$$

  Amb aquests blocs, les diferències poden calcular respecte del primer valor del bloc de manera que el vector resultant *a* sempre té un nombre de elements $T - 1$.

  $$a = ((x_0 - x_1), (x_0 - x_2), (x_0 - x_3))$$

  Així, el vectors de diferències resultants serien:

  $$a_0 = (1, 2, 0)$$
  $$a_1 = (1, -1, 0)$$
  $$a_2 = (-2, -1, -1)$$

  Es pot veure que els valors resultants són nombres sencers, la qual cosa permet estudiar les diferències en valors absoluts i no com referències entre els valors locals de cada bloc.

Si s'agrupen tots els vectors de diferències el resultat és una matriu *A* de dimensions N^{T-1}, on $N \in \mathbb{N} > 0$ representa el nombre de possibles vectors, el qual està directament relacionat amb el valor màxim de diferència entre els elements dels blocs com es veurà a continuació.

$$A = \begin{pmatrix} a_0 & \cdots & a_N \\ \vdots & \ddots & \vdots \\ a_{N^{T-2}} & \cdots & a_{N^{T-1}} \end{pmatrix}$$

- El valor màxim de diferència *S*.

  Pel cas de l'esteganografia en imatges, el valor màxim està acotat per la definició del píxel. En canvi, no hi ha cap acotació possible pels grups de PDU pel protocol RTP, ja que aquest només transmet el que l'aplicació li demana.

  Així, per exemple, si un dels extrems a una conversa decideix habilitar la seva càmera, no hi haurà PDU de vídeo a la conversa en un dels sentits. Per tant, no es pot calcular a priori un valor de *S*.

  No obstant això, analitzant el tràfic resultant de converses sense aplicar l'algorisme d'ocultació i enviant àudio i vídeo, es va veure que els valors dels grups de les PDU són nombres petits, la qual cosa vol dir que s'envien PDU dels dos tipus de forma contínua perquè la conversa pugui ser fluida.

  Atès que els valors dels grups no eren molt grans, si es fixa $S = 5$ i tenint en compte que les diferències poden ser nombres negatius, queden acotades als valors:

  $$\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$$

  Així, totes les diferències superiors a $|S|$ passen a agafar els valors $-S$ o S. És a dir, el nombre de possibilitats, definides a la matriu *A* per *N*, és $2S + 1$.

$$A = \begin{pmatrix} a_0 & \cdots & a_{(2S+1)} \\ \vdots & \ddots & \vdots \\ a_{(2S+1)^{T-2}} & \cdots & a_{(2S+1)^{T-1}} \end{pmatrix}$$

Es pot veure que com més gran sigui *T,* més relacions entre grups s'estudien. Això permet veure el tràfic d'una forma més global atès que s'avaluen les relacions entre més grups de PDU. No obstant això, per un valor fixo de *S*, la variació de *T* fa que el nombre de possibilitats creixi de forma exponencial.

En canvi si es fixa *T*, com més gran sigui *S* més informació sobre el comportament de la xarxa s'analitza, és a dir, més informació de la

distribució del tràfic de la xarxa. El nombre de possibilitats creix de forma lineal.

Per avaluar els nombres adients de les dues variables, cal tenir present quines són les *features* que es passaran als algorismes de *machine learning*. Les característiques són uns valors numèrics agrupats en vectors que es passen als algorismes de classificació etiquetats de forma que puguin establir el llindar oportú. Així, cal que els vectors tinguin la mateixa grandària i que totes les posicions representin el mateix concepte.

En el cas d'aquest projecte, el que resulta més interessant és comparar les distribucions estadístiques resultants per a grups de diferències limitats. Això és analitzar les ocurrències que hi ha de cada una de les possibilitats de les diferències a la matriu de diferències totals *A*, i aquests nombres resultants són les *features*.

Per tant, si el tràfic que es vol estudiar genera els vectors de diferències *c* tales que:

$$c = (c_{-x},\ \cdots\ ,\ c_x), \qquad \forall x\ \epsilon\mathbb{Z} \mid x \in (-S, S)$$

On el nombre d'elements de *c* és (*T-1)*.

El conjunt que agrupa tots els vectors que sorgeixen del tràfic és la matriu *C* amb unes dimensions que depenen de la quantitat de tràfic analitzat però que com màxim pot ser $(2S+1)^{T-1}$.

$$C = \begin{pmatrix} c_0 \\ \vdots \\ c_{(2S+1)^{T-1}} \end{pmatrix} = \begin{pmatrix} c_0 & \cdots & c_{(2S+1)} \\ \vdots & \ddots & \vdots \\ c_{(2S+1)^{T-2}} & \cdots & c_{(2S+1)^{T-1}} \end{pmatrix}$$

Les *features* són el nombre de cada vector de *C* que hi ha a la matriu *A*, és a dir:

$$\forall c \in C \mid n(c \in A)$$

Dit d'una altra forma, la matriu *C* és un subconjunt de la matriu *A*.

$$C \subset A$$

Perquè les característiques resultants siguin eficients per obtenir un bon classificador, tots els vectors de diferènciés possibles, agrupats en la matriu *A*, han de tenir una ocurrència a la matriu de vectors de diferències *C* que ha donat com a resultat el tràfic analitzat.

$$\forall a \in A \mid n(a \in C) > 0$$

O, cosa que és el mateix:

$$A \subset C$$

Per tant, si els valors de *T* i *S* són molt grans, és molt més probable que no es donin aquestes relacions i els algorismes no entrenin correctament. A l'apartat de les proves i resultats es donaran els valors concrets que s'han provat i els resultats obtinguts.

Una vegada triats els valors de *T* i *S* si $A \subset C$, les característiques formaran un vector D de dimensions $1\ x\ (2S+1)^{T-1}$.

$$D = \begin{pmatrix} n(c_0 \in A) \\ \vdots \\ n(c_{(2S+1)^{T-1}} \in A) \end{pmatrix}^t = (n(c_0 \in A), \dots, n(c_{(2S+1)^{T-1}} \in A))$$

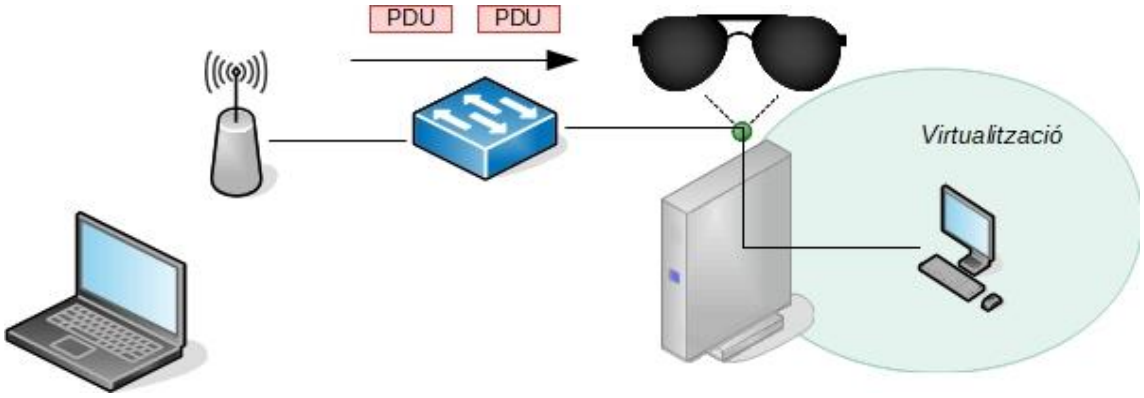
Cal tenir present que els tipus de PDU als quals pertanyen els valors no resulten interessants, ja que d'aquesta forma d'aproximació al mètode permet que es pugui aplicar a qualsevol tràfic de xarxa amb un *network covert channel* que utilitzi la tècnica d'alteració de l'ordre de les PDU per a dos tipus de PDU diferents.

5.3 Proves i resultats

Atès a la importància que té aquest punt, s'ha decidit estructurar-ho en tres apartats. Al primer es descriu com s'han fet les proves, al segon l'entrenament de l'algorisme de *machine learning* i al tercer el resultats de les proves amb tràfic nou en temps real.

5.3.1 Infraestructura

Per aconseguir escoltar el tràfic de una forma eficient, cal introduir el detector en mig del tràfic. A la literatura, aquesta solució es sol basar en la publicació de Simmons [67] coneguda com el *problema del presoner*: siguin dos presoners, Bob i Alice, els quals tenen dret de escriure un *missatge* per passar-li a l'altre però el guarda que trasllada el paper, Wendy, pot llegir la conversa. La història serveix per explicar que els dos presoners han d'utilitzar un codi secret per emmascarar els missatges que no vulguin que *Wendy* entengui.
Pel que fa al projecte, la idea és introduir el programari NSD com si fos *Wendy* amb un tercer dispositiu dins l'escenari de proves descrit al punt 4.1.3.2. És important pensar que la ubicació del guarda no és realment crítica en aquest escenari atès que la pèrdua de PDU afectarà als dos interessats, el guarda i el receptor del missatge.



Imatge 5.05. Infraestructura local.

5.3.2 Entrenament de l'algorisme de *Machine learning*

L'entrenament no es va fer en temps real sinó que primer es va capturar el tràfic amb el programa *Wireshark* i es va carregar amb el programa *detector* a la base de dades discriminant cada fitxer resultant de la captura com tràfic normal o amb canals encoberts.

De les dues categories es varen aconseguir conjunts de dades semblants d'aproximadament quatre milions i mig de PDU (ja que només es va capturar el tràfic que resultava interessant per al projecte). El tràfic es va capturar al mateix escenari de les proves fent una videotrucada utilitzant un canal de televisió com font d'àudio i so. Amb aquest volum, varen sortir uns 900.000 grups de PDU de cada categoria, els quals varen portar a 899.800 vectors de diferències en nombres sencers. Aquesta base de dades no forma part de la memòria per raons d'espai, ja que ocupa poc més de 11GB.

Amb aquestes dades es va començar a fer feina amb la següent seqüència d'accions.

1) Escollir els valors de *T* i *S*.

   Segons aquests valors varien les característiques que es passen als algorismes d'aprenentatge automàtic. S'han provat els següents valors:

   - $T = [3, 4, 5]$. És a dir, les llargàries dels blocs provats són de tres, quatre o cinc grups de PDU. S'han trobat diferències significatives en el temps de preparació que requereix formar tots els blocs possibles i fer els recomptes al tràfic. Com s'ha explicat, com més gran més tràfic es requereix per omplir totes les possibilitats i això repercuteix en la capacitat de detecció. Per a $T = 5$ va estar aproximadament quatre hores per trobar valors per omplir tota la matriu resultant en molt poques matrius per entrenar l'algorisme de *machine learning*. Atès que els

objectius són que tots els vectors de possibilitats tinguin resultats i que el classificador pugui funcionar en temps real, s'ha triat el valor $T = 3$.

- $S = [2, 3, 4, 5]$. Com es va explicar, com més gran és el valor màxim de diferències més possibilitats a la matriu resultant de *features*. De la mateixa forma que per la variable anterior, com més possibilitats hi ha, més tràfic s'ha de capturar per assegurar que no hi hagi possibilitats sense ocurrències ja que els algorismes de *machine learning* no fan bones prediccions si es dóna aquesta situació. La conseqüència de necessitar més tràfic és que només es podrien detectar grans missatges ocults. Per tant, el valor triat per assolir els objectius és $S = 2$.

Per tant, a la matriu de possibilitats hi ha $(2S + 1)^{T-1} = (2 * 2 + 1)^{3-1} = 25$ vectors, és a dir, quedaria tal que així:

$$A = \begin{bmatrix} [-2,-2] & [-1,-2] & [0,-2] & [1,-2] & [2,-2] \\ [-2,-1] & [-1,-1] & [0,-1] & [1,-1] & [2,-1] \\ [-2,0] & [-1,0] & [0,0] & [1,0] & [2,0] \\ [-2,1] & [-1,1] & [0,1] & [1\ 1] & [2,1] \\ [-2,2] & [-1,2] & [0,2] & [1,2] & [2,2] \end{bmatrix}$$

I el vector de *features* el següent:

$$D = (n([-2,2] \in C), \dots, n([2,2] \in C))$$

2) Escollir el nombre d'elements de la matriu *C*.

Una vegada que és coneguda la matriu de possibilitats, cal triar el nombre de vectors de diferències que formen part de la mateixa "conversa" que formarà un vector de *features*.

El raonament és semblant al punt anterior: com més gran sigui el nombre d'elements de *C*, es pot dir que més resolució tindran les característiques però també es requerirà més tràfic per treure conclusions fiables.

Per tant, és necessari enfocar-ho d'una altra forma. Com més grups de diferències a cada conversa, més gran ha de ser el missatge ocult i això no és interessant.

La relació entre el nombre de *features F* i el nombre de grups de diferències *M* es basa en el nombre de la grandària de bloc *T*.

$$F = M - (T - 1)$$

Així, per exemple, per formar converses de $F = 1000$ *features* es requereixen $M = F + (T - 1) = 1002$ grups de diferències. Si cada

grup de diferències representa 1 bit, 1002 grups de diferències equivalen a 125 bytes. És a dir, es requeririen 125 lletres en cas de ser missatges en text pla.

Encara que podria ser una mida apropiada, s'ha decidit anar un poc més lluny i es mostraran també els resultats amb proves per a $F = 100$ per dues raons importants:

1. Demostrar que es poden detectar missatges molt curts, ja que $F = 100$ equival a la transmissió de 13 bytes.

2. L'anàlisi estadístic s'havia fet sota aquest nombre de grups de PDU i, per tant, és la millor forma d'avaluar el principi de indetectabilitat.

Amb tots els paràmetres escollits es pot veure tot el que aporta la tècnica PPD quan s'estudien les relacions entre els grups de PDU al flux de tràfic RTP. A la imatge 5.06 es pot veure la relació entre dos grups de característiques dels dos tipus de tràfics: en verd el que porta un missatge ocult i en vermell el que no.

La gràfica de la dreta és la distribució del tràfic on cada posició representa un vector de diferències de la matriu *A*. Així, es pot veure que el tràfic transmet per la xarxa tal qual l'envia l'aplicació té dues característiques fonamentals:

- Moltes posicions de vectors de la matriu no cobertes.

  $$a = [N, M] = 0 \qquad \forall N, M \in \mathbb{Z} \mid N, M \in (-S, S)$$

  És a dir, que els blocs amb els nombres de grups de PDU no donen totes les possibilitats de diferències de la matriu *A*. Es pot dir, per tant, que pel que fa al tràfic sense missatges ocults:

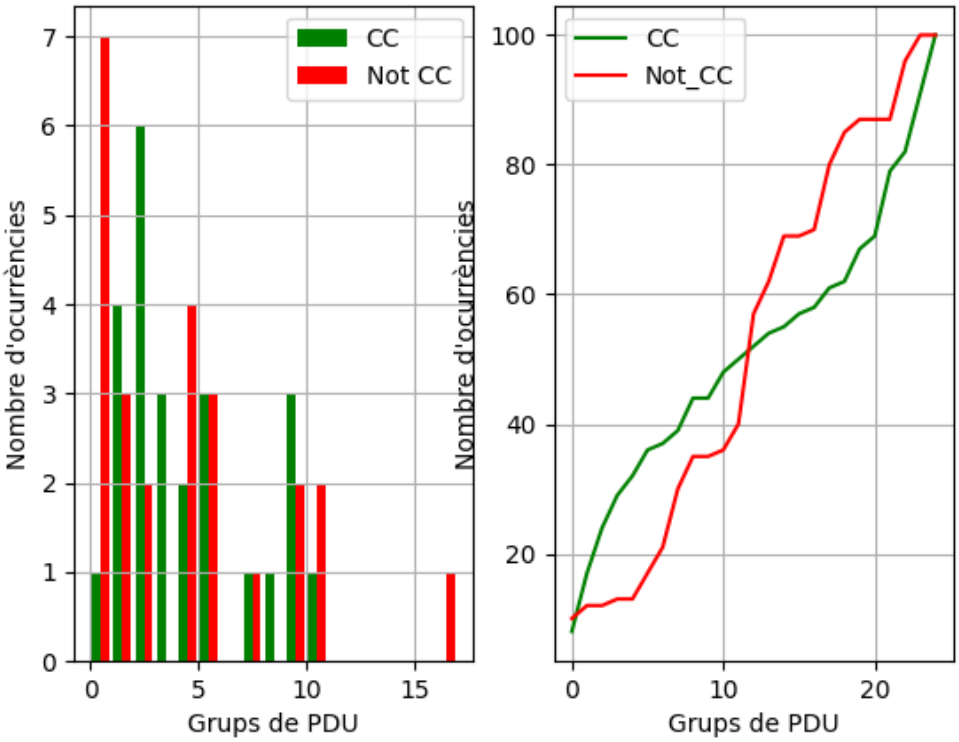
  $$A \notin C$$

- Hi ha una posició amb moltes ocurrències i no és un altre que $a = [0, 0] = 17$. És a dir, si no hi ha missatges ocults, molts grups de PDU s'envien amb el mateix nombre de PDU.

En canvi, si hi ha un missatge ocult, els vectors de diferències poblen totes les possibilitats i es reparteixen per tot el conjunt *A*.

La gràfica de l'esquerra presenta dues corbes cumulatives molt diferenciades i representatives: per una banda, el tràfic portador d'un *network covert channel* fa un corba molt suau que té gairebé forma sigmoide girada, la qual cosa vol dir les diferències produïdes es distribueixen al llarg del vector de característiques però es concentren als límits. En canvi, el tràfic normal presenta uns

escalons molt marcats en representació de la concentració dels grups a posicions contigües del vector de *features*.



Imatge 5.06. Distribucions dels vectors de *features*.

Els dos vectors que han donat aquests resultats són els de la figura 5.07.

```
cc = [8, 9, 7, 5, 3, 4, 1, 2, 5, 0, 4, 2, 2, 2, 1, 2, 1, 3, 1, 5, 2, 10, 3, 9, 9]
ncc = [10, 2, 0, 1, 0, 4, 4, 9, 5, 0, 1, 4, 17, 5, 7, 0, 1, 10, 5, 2, 0, 0, 9, 4, 0]
```

Imatge 5.07. Vectors de diferències dels gràfics de la imatge 5.06.

3) Entrenament dels algorismes segons les *features* donades.

Per raons de llargària no es reflectiran totes les proves per entrenar els classificadors però a la següent taula es veuran algunes de les quals es varen entrenar i les principals variables que es varen modificar.

Algorisme	Tipus	Paràmetres
ExtraTreesClassifier	*Arbre de decisió*	*n_estimators: [1000, 2000, 3000, 4000]* *criterion: [gini, entropy]* *random_state=[1, 2, 3, 4, 5]*

RandomForestClassifier	*Arbre de decisió*	*n_estimators: [1000, 2000, 3000, 4000], criterion: [gini, entropy] random_state=[1, 2, 3, 4, 5]*
AdaBoostClassifier	*Arbre de decisió*	*base_estimator: [ExtraTreesClassifier(max_depth: [10, 50, 100]), RandomForestClassifier(max_depth: [10, 50, 100])] n_estimators: [100, 1000, 2000, 3000, 5000, 10000] learning_rate: [10, 1, 0.5, 0.1, 0.01, 0.001]*
SVC	*SVM*	*gamma: [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001] C: [1, 10, 100, 1000, 10000, 100000, 1000000] kernels: [rbf, sigmoid]*
MLPClassifier	*Xarxes neuronals*	*solver: lbfgs max_iter: [8000, 9000, 10000, 15000, 20000] alpha: [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001] hidden_layer_sizes: [10, 50, 100, 1000] random_sate: [1, 2, 3, 4, 5]*

Taula 5.03. Algorismes entrenats.

Cal destacar que els paràmetres els ha anat modificant de forma automàtica la funció *GridSearchCV* [68] i que el format d'entrenament ha estat per a tots igual: un conjunt del 80% de les *features* per entrenar i un 20% per avaluar.

4) Elecció en funció de les mètriques.

Al llarg de les proves, s'ha vist que les mètriques *F1-score* i MCC es projecten de forma paral·lela de manera que totes dues alteren els valors en el mateix sentit.

Cal dir que tots els algorismes varen tenir resultats molt semblants, la qual cosa s'interpreta com que el mèrit és l'adaptació de la tècnica PPD al tràfic de xarxa. Els únics mètodes que han donat uns resultats més diferents han estat els basats a minimitzar una funció d'error mitjançant un gradient (*GradientBoostingClassifier*) i per això no s'han inclòs a la memòria.

Els resultats es mostren a la taula 5.04 amb els paràmetres de configuració dels algorismes i segons el nombre de *features F* que, com s'ha explicat, equival al nombre de bits necessari per treure conclusions. S'ha ressaltat el millor resultat obtingut i les seves característiques.

Algorisme	Paràmetres	*F*	
		100	1000
ExtraTreesClassifier	*n_estimators: 1000* *criterion: gini* *random_state= 2*	*F1-score: 0.9504736129905278* *MCC: 0.9001084218654885*	*F1-score: 0.968339483394834* *MCC: 0.9356016041221814 9*
RandomForestClassifier	*n_estimators: 2000* *criterion: gini* *random_state=1*	*F1-score: 0.9485373781148428* *MCC: 0.8959297864552106*	*F1-score: 0.9764381360471344* *MCC: 0.9404083919874 3573*
AdaBoostClassifier	*base_estimator: RandomForestClassifier* *(max_depth: 10)* *n_estimators: 10000* *learning_rate: 0.1*	*F1-score: 0.9572421165152325* *MCC: 0.9123106503303196*	*F1-score 0.9917355371900827* *MCC 0.9834699358669274*
SVC	*gamma: 0.001* *C: 100* *kernels: rbf*	*F1-score: 0.9523821246989563* *MCC: 0.9013116723417146*	*F1-score: 0.982578548212351* *MCC: 0.96856840465218 72*
MLPClassifier	*solver: lbfgs* *max_iter: 8000* *alpha: 0.1* *hidden_layer_sizes: 12* *random_sate: 2*	*F1-score 0.9514564183835182* *MCC: 0.9000438820846982*	*F1-score: 0.9740308795147505* *MCC: 0.9546517671131 6741*

Taula 5.04. Millors algorismes entrenats.

Per tant, el millor classificador és l'*AdaBoost* que es basa a resultats obtinguts d'algorismes simples per trobar el millor resultat. És a dir, el que fa és entrenar els classificadors de forma iterativa amb les dades que varen resultar errònies de l'anterior iteració. En aquest cas, el classificador dèbil és un *RandomForestClassificator* el qual genera nombrosos arbres de decisió amb conjunts de *features* (no amb totes) i als resultats se'ls assigna un vot. El que tingui el vot major, és el resultat de l'arbre.

A la taula 5.04 es pot veure com, clarament, amb una llargària superior el resultat és molt millor també per tots els algorismes. Es podria pensar a

utilitzar vectors de *features* molt més llargs però caldrien volums de tràfic molt elevats per entrenar el classificador i per analitzar el tràfic en temps real. Això pot provocar que si el volum d'informació que es transmet de forma oculta és petit, les característiques bé agafarien tràfic normal que amagarien el missatge o bé no hi hauria prou tràfic per generar les *features* requerides pel classificador.

En tot cas s'ha provat d'entrenar un classificador amb l'algorisme i els paràmetres esmentats per a $F = 64$, és a dir, per veure si es pot detectar amb fiabilitat paraules de 8 caracters. El resultat és el següent:

F = 64
F1-score: 0.9293742660627411
MCC: 0.8534624318801631

Taula 5.05. Resultats per a *F=64*.

Es pot veure com a mesura que es baixa el nombre de característiques, als algorismes les costa més donar resultats però, no obstant això, un 0,85 a la MCC i un 92,9% al *F1-score* es consideren un molt bon resultat atès que hi ha molt poques mostres per analitzar.

La darrera prova era amb tràfic a l'entorn real de la xarxa local que s'ha utilitzat al llarg de tot el treball. El que s'ha fet és muntar l'escenari per alterar l'ordre de les PDU segons s'ha descrit i analitzar el tràfic de la interfície amb el programa *detector* que havia carregat el classificador en memòria.

```
2020-06-01 19:18:09,650 - bin.Monitor - INFO - ------------------------ PACKET LIVE REPORT ------------------------
2020-06-01 19:18:09,651 - bin.Monitor - INFO - -- Read:
2020-06-01 19:18:09,651 - bin.Monitor - INFO -    TCP|UDP|RTP|ICMP|ERROR|TOTAL: 65|3600|6|51|3722
2020-06-01 19:18:09,651 - bin.Monitor - INFO - -- Inserted into the database:
2020-06-01 19:18:09,651 - bin.Monitor - INFO -    TCP|UDP|RTP|ICMP|ERROR|TOTAL: 65|3001|600|0|3666
2020-06-01 19:18:09,651 - bin.Monitor - INFO - -- Conversations identified:
2020-06-01 19:18:09,651 - bin.Monitor - INFO -    TCP|UDP|RTP|ICMP|TOTAL: 0|0|3|0
2020-06-01 19:18:09,651 - bin.Monitor - INFO -
2020-06-01 19:18:14,743 - bin.Flow - DEBUG - Flow: RTP: flow id 5ed5371d056aac77495af64b, category 2
2020-06-01 19:18:14,745 - bin.Machine_Learning - INFO - Getting features for flow '5ed5371d056aac77495af64b'..
2020-06-01 19:18:18,856 - bin.Machine_Learning - INFO - Got features for flow '5ed5371d056aac77495af64b'
2020-06-01 19:18:18,856 - bin.Machine_Learning - INFO - Evaluate: predicting for id '5ed5371d056aac77495af64b'..
2020-06-01 19:18:21,217 - bin.Machine_Learning - CRITICAL - NETWORK COVERT CHANNEL! With 0.9071635806588856 MCC metric, Source
IP 192.168.22.155 and Destination IP 192.168.22.175, with RTP
```

Imatge 5.07. Detecció en temps real.

Tal com es pot veure a la figura 5.07, el programa *detector* treu un missatge de la categoria *CRITICAL* quan detecta una conversa que ha trobat que porta un missatge ocult. D'aquesta imatge cal dir que al mode DEBUG l'eina treu la informació de com el flux de tràfic transita per l'aplicació. Primer es treu una llista dels fluxos que hi ha a la base de dades i aquells que siguin del protocol RTP, es van analitzant.

L'aplicació està implementada per suportar els següents dos casos:

- No hi ha tràfic suficient per treure les *features*: surt un missatge però no es descarta el flux.

- El missatge és molt petit i només s'envia durant una part de la videotrucada de forma que quedi amagat amb el tràfic normal. Com s'ha explicat, amb un nombre de característiques tant petit com cent, qualsevol intent d'enviar més de 12 bytes serà interceptat. Aquesta prova es va realitzar amb èxit iniciant una videotrucada i enviant el missatge ocult quan ja estava iniciada donant com a resultat un positiu.

No obstant això, té una mancança important: no s'elimina el tràfic ja comprovat i caldria fer-ho de qualsevol de les següents formes: de tot el flux d'una conversa o del tràfic comprovat dins la conversa. Això permetria gestionar grans volums de tràfic però requereix molt més temps de desenvolupament.

Per una altra banda, s'ha generat tràfic sense portar missatges ocults i el resultat ha estat també positiu, ja que el programa no ha mostrat cap alarma. A la imatge 5.08 es mostra com el programa no treu cap missatge encara que hi ha una videotrucada en marxa. Es pot veure a la imatge 5.09 els fluxos emmagatzemats a la base de dades als quals hi ha un registre pel tràfic entre les dues IP de la xarxa local que fan la videotrucada. Es veuen més registres que fluxos ha identificat l'aplicació perquè cal recordar que és tràfic UDP i només s'analitza el tràfic RTP (que va sobre el protocol UDP). Cal explicar que es veuen tres fluxos entre les dues IP involucrades en la videotrucada perquè en una conversa SIP hi ha més protocols involucrats que no s'han explicat perquè no aportaven res al projecte.

```
patrik@CaNostra:~/PycharmProjects/detector$ sudo python3 detector.py -d
2020-06-01 19:23:05,529 - bin.Init - INFO - Starting RTP Flows process..
2020-06-01 19:23:21,613 - bin.Machine_Learning - ERROR - Flow '5ed53981ef385512eeb1e3b3' does not have enough traffic.
2020-06-01 19:23:48,729 - bin.Machine_Learning - ERROR - Flow '5ed5399222507aaa98b1e5e8' does not have enough traffic.
```

Imatge 5.08. Tràfic sense missatge ocult.

```
{ "_id" : { "Source_IP" : "192.168.22.175", "Dest_IP" : "192.168.22.155", "id" : ObjectId("5ed5399966065c0da7b1e6c2") } }
{ "_id" : { "Source_IP" : "192.168.22.175", "Dest_IP" : "192.168.22.155", "id" : ObjectId("5ed5398166065c0da7b1e39f") } }
{ "_id" : { "Source_IP" : "192.168.22.175", "Dest_IP" : "192.168.22.155", "id" : ObjectId("5ed539a666065c0da7b1e887") } }
{ "_id" : { "Source_IP" : "84.17.62.194", "Dest_IP" : "192.168.22.100", "id" : ObjectId("5ed539ee66065c0da7b1e9ef") } }
```

Imatge 5.09. Registres a la base de dades.

Les imatges 5.08 i 5.09 validen que el detector no avisa si el tràfic no porta un missatge ocult. Ara bé, durant les proves s'ha vist que si s'inclouen els primers 5 grups de les videotrucades, totes les converses donen positiu. Això és perquè al començament el tràfic s'ha de regular i s'envien aquests

grups amb un nombre de PDU molt elevades (el primer grup és d'àudio i sol sobrepassar les 50 PDU, el segon, de vídeo, les 20). La versió del programa presentat agafa totes les PDU.

Es varen fer proves amb 1000 característiques per vector i no hi havia problema amb els primers grups de PDU de les videotrucades. S'interpreta que amb més grups de PDU la tècnica PPD remarca més les diferents distribucions i, per tant, és més fàcil distingir entre els tipus de tràfic.

6. Conclusions

Al llarg dels capítols 2 i 3 s'han descrit els fonaments teòrics del projecte, els quals s'han utilitzat al capítol 5 a l'hora de desenvolupar les proves realitzades i els resultats pràctics obtinguts. El capítol 4 ha servit per mostrar les eines desenvolupades i exemples del seu funcionament.

L'objectiu principal era dissenyar i implementar un programari funcional que servís com detector d'una tècnica concreta d'esteganografia en xarxa anomenada "alteració de l'ordre" i basat en algorismes de *machine learning*. En aquest sentit el resultat ha estat molt satisfactori, com s'ha vist al punt 5.4.

L'objectiu paral·lel era que el detector es basés en un algorisme concret per crear un *network covert channel* a partir de tràfic VoLTE. La impossibilitat d'obtenir tràfic real va portar a dissenyar l'eina presentada al punt 4.1.1. Encara que es va descartar, els processos de disseny i implementació i els resultats obtinguts van resultar un molt interessants, la qual cosa va decidir la inclusió del procés de forma descriptiva a la memòria.

Una vegada descartat el simulador, es va afrontar amb decisió el procés de dissenyar un algorisme utilitzant un protocol real. Aquesta part va suposar un dels canvis realitzats per les raons exposades als punts corresponents però el resultat va ser molt satisfactori perquè compleix el que s'anomena principi de indetectabilitat (vegeu 5.2) segons els estudis esmentats al llarg de la memòria.

Tots els processos creatius de disseny i implementació dels diferents punts han servit per aprendre sobre *network covert channel* i *machine learning*. Amb els coneixements adquirits en aquest treball, l'evolució del programari i d'investigacions futures és un camí que caldrà una reflexió.

El punt a millorar del projecte és probablement la creació i presentació de diferents escenaris de proves per acabar de comprovar la detecció per una part i per donar robustesa al canal de comunicació encobert. La dificultat afegida de la situació social durant aquest 2020 ha compromès

molt el temps de dedicació, el qual hagués servit per fer proves més exhaustives. A més a més, tal com s'ha comentat a l'apartat 4.2.4, caldria una millora del codi de l'eina que detecta els *network covert channels* atès que té problemes per gestionar grans volums de tràfic.

Respecte a evolucions futures, a part de millorar els punts esmentats, hi ha dos grans possibilitats:

- Desenvolupar el detector amb mòduls nous que serveixin per detectar més tipus de *network covert channels* i que analitzi més protocols.

- Estudiar les possibilitats del mètode PPD al món de l'esteganografia en xarxa. Aquest treball ha servit per demostrar que és un mètode d'extraure *features* molt potent que podria servir per detectar mètodes del tipus *storage* com aleshores els d'alteració de camps reservats o no utilitzats atès que al tràfic normal solen tenir uns valors de zero per a cada bit. És clar que la part interessant seria la seva aplicació als de tipus *timing* però també requereix més temps d'estudi i anàlisi de cara a obtenir unes característiques que siguin un factor clau, com ha passat en aquest projecte.

7. Glossari

- **Arbre de decisió**: model de l'aprenentatge automàtic que pren les decisions de forma lògica per nodes agrupats en cascada.
- **Alteració de l'ordre**: tècnica per encobrir un missatge a un flux de xarxa que modifica la disposició de les PDU de forma deliberada.
- **Esteganografia**: terme que engloba totes les tècniques que serveixen per ocultar informació dins altres. Poden ser al mon físic o al món digital.
- **Estegoanàlisi**: totes les tècniques que analitzen les tècniques d'esteganografia a fi de trobar una forma de detectar o obtenir la informació oculta.
- **Flux de tràfic**: fa referència al tràfic RTP que transita entre els interlocutors sense que es distingeixi el tipus de contingut multimèdia que porta cada PDU.
- **Grup de PDU**: aplicat a l'obtenció de característiques, es diu del nombre de PDU del mateix tipus, ja siguin àudio o vídeo, que s'envien de forma consecutiva.
- **IP**: protocol d'enrutament utilitzat a la xarxa Internet.
- **KLD**: Divergència de Kullback-Leiber, serveix per obtenir una mesura sobre la distància entre dues funcions de probabilitat.
- **KLS**: test de Kolmogorov-Smirnov, s'utilitza per comparar la similitud entre dues funcions de distribució.
- **Linphone**: programari que permet realitzar videotrucades segons el protocol VoIP.
- ***Machine learning***: algorismes de la intel·ligència artificial que milloren mitjançant les experiències.
- ***Network covert channel***: terme que engloba totes les tècniques existents per amagar informació al tràfic de xarxa.

- ***Network covert storage channel***: categoria de *network covert channels* basats a explotar el contingut de qualsevol camp de les PDU.

- ***Network covert timing channel***: categoria de *network covert channels* basats a alteracions al flux de xarxa.

- **NFQUEUE**: mòdul del nucli de GNU/Linux que permet capturar tràfic de xarxa.

- **PCAP**: *Packet CAPture*, format dels fitxers resultants de la captura de PDU a una interfície per part del programari anomenat de la mateixa forma.

- **PDU**: *Protocol Data Unit*, és cada element individual i autònom que transmet informació entre entitats del mateix nivell a una xarxa de telecomunicació.

- **pip**: gestor de paquets o programaris de *Python*.

- **PPD**: *Patterns of Pixels Differences*, és una tècnica per extraure característiques basades en la relació amb els elements consecutius.

- **RTP**: protocol de xarxa del nivell quatre per transmetre contingut multimèdia dins una videotrucada utilitzat per VoIP.

- **S**: és un paràmetre que s'usa per aconseguir les característiques segons la adaptació de la tècnica PPD i que acota el valor màxim de diferència que pot haver entre dos grups de PDU.

- **Sniffer**: programari que captura el tràfic d'una interfície de xarxa.

- **SVM**: *Support Vector Machine*, model de l'aprenentatge automàtic basat en calcular la distància entre les mostres més properes de diferents tipus.

- **T**: és un paràmetre utilitzat per l'obtenció de les característiques segons la tècnica adaptada PPD i que mesura el nombre de grups de PDU que formen part de cada bloc.

- **TCP**: protocol de xarxa del nivell de transport que permet garantir la arribada de la informació a l'altre extrem.

- **TCP/IP**: pila de protocols de les xarxes de telecomunicació que s'utilitza a la xarxa Internet.

- **UDP**: protocol de xarxa de nivell quatre caracteritzat per no garantir la arribada de la informació a l'altre extrem.

- **VoIP**: conjunt de tecnologies que permeten enviar tràfic de veu i vídeo sobre la xarxa Internet.

- **VoLTE**: *Voice over LTE*, permet realitzar videotrucades sobre la pila de protocols LTE, coneguts també com 3.95G.

- **Xarxes neuronals**: model de l'aprenentatge automàtic caracteritzat per simular un conjunt de neurones relacionades per capes.

- **Wireshark**: programari que captura i interpreta el tràfic d'una interfície de xarxa.

8. Bibliografia

1. Serra, J. i Lerch, D., *Esteganografia y estegoanálisis*, ZEROXWORD COMPUTING, Móstoles, 2014.

2. https://www.urosario.edu.co/Revista-Nova-Et-Vetera/Vol-1-Ed-2/Columnistas/Apuntes-sobre-la-Esteganografia-de-Trithemius/, vist Maig 2020.

3. http://atalap.blogspot.com/2007/09/història-de-la-criptografa-y.html, vist Febrer 2020.

4. B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, vol. 16, no. 10, pp. 613-615, 1973.

5. Xiaosong Zhang, Liehuang Zhu, Xianmin Wang, Changyou Zhang, Hongfei Zhu, Yu-an Tan. A packet-reordering covert channel over VoLTE voice and video traffics. *Journal of Network and Computer Applications*, 26 (2019) 29–38.

6. S. Wendzel, S. Zander, B. Fechner, and C. Herdin. Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p.50, 2015.

7. S. Wendzel. Protocol-independent Detection of “Messaging Ordering” Network Covert Channels. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES ’19)*, August 26–29, 2019, Canterbury, United Kingdom. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3339252.3341477

8. Cachin C. An information-theoretic model for steganography. In *Proceedings of Second International Workshop on Information Hiding*, USA, 2004, pp. 306–318.

9. Muawia A. Elsadig, Yahia A. Fadlalla. Network Protocol Covert Channels: Countermeasures Techniques. *9Th IEEE-GCC Conference and Exhibition (GCCCE)*, 2017.

10. Pradhumna Lal Shrestha, Fahimeh Rezaei and Hamid Sharif. A Support Vector Machine-Based Framework for Detection of Covert Timing Channels. *IEEE Transactions on dependable and secure computing*, VOL. 13, NO. 2, MARCH/APRIL 2016

11. *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermesaures*, First Edition. Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. 2016 by The Institute of Electrical and Electronics Engineers, Inc. Published 2016 by John Wiley & Sons, Inc.

12. J. Lubacz, W. Mazurczyk, and K. Szczypiorski. Principles and overview of network steganography. *IEEE, Communications Magazine*, 52(5):225-229, 2014.

13. S. J. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In *Proceedings of the Information Hiding Conference 2005*, Vol 3727 of Lecture Notes in Computer Science, pp. 247-261. Springer, 2005

14. C. G. Girling. Covert channels in LAN's. *IEEE Transactions on Software Engineering*, 13(2):292-2996, 1987.

15. A. Dyatlov i S. Castro. Exploitation of data streams authorized by a network access control system for arbitrary data transfers: tunneling and covert channels over the http protocol. *Technical Report, Gray-World.net*, 2005.

16. R. Rios, J. Onieva, i J. Lopez. HIDE_DHCP: covert communications through network configuration messages. In *Proceedings of the IFIP TC 11 27th International Information Security Conference*. Springer, 2012.

17. X. Zou, Q. Li, S. Sun, i X. Niu. The research on information hiding based on command sequence of FTP protocol. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, pp. 1079-1085, September 2005.

18. Z. Trabelsi and I. Jawhar. Covert file transfer protocol based on the IP record route option. *Journal of Information Assurance and Security*, 5(1):64-73, 2010.

19. S. Bellovin. RFC 1948: defending against sequence number attacks. https://www.ietf.org/rfc/rfc1948.txt, Maig 2020.

20. N. Lucena, J. Pease, P. Yadollahpour, and S. J. Chapin. Syntax and semantics-preserving application-layer protocol steganography. In *Proceedings of the 6th Information Hiding Workshop*, May 2004.

21. N. Lucena, G. Lewandowski, and S. Chapin. Covert channels in IPv6. In *Proceedings of the 5th International Workshop on Privacy Enhancing*

Technologies (PET '05), Vol. 3856 of Lecture Notes in Computer Science, pp. 147-166. Springer, 2006.

22. S. Zander, G. Armitage, and P. Branch. Covert channels in the IP time to live field. *Australia Telecommunication Networks and Applications Conference (ATNAC '06)*, pp. 298-302, 2006.

23. M. Wolf. Covert channels in LAN protocols. In *Proceedings of the Workshop on Local Area network Security (LANSEC)*, pp. 91-101, 1989.

24. M. A. Padlipsky, D. W. Snow, and P. A. Karger. Limitations of end-to-end encryption in secure computer networks. *Technical Report*, ESD-TR-78-158, Mitre Corporation, Auguts 1978.

25. V. Berk, A. Giani, and G. Cybenko. Detection of convert channel encoding in network packet delays. *Technical Report* TR2005-536, Department of Computer Science, Dartmouth College, November 2005. http://www.ists.dartmouth.edu/library/149.pdf

26. M. H. Kang and I. S. Moskowitz. A pump for rapid, reliable, secure communication. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pp. 119-129, 1993.

27. S. D. Servetto and M. Vetterli. Communication using panthoms: covert channels in the internet. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, June 2001.

28. C. Kräzter, J. Dittmann, A. Lang, and T. Kühne. WLAN steganography: a first practical review. In *Proceedings of the 8th ACM Multimedia and Security Workshop*, September 2006.

29. R. C. Chakinala, A. Kumarasubramanian, R. Manokaran, G. Noubir, C. P. Rangan, and R. Sundaram. Steganographic communication in ordered channels. In *Proceedings of the 8th International Workshop on Information Hiding*, pp. 42-57, July 2006.

30. T. Handel and M. Sandford. Hiding data in the OSI network model. In *Proceedings of the First International Workshop on Information Hiding*, pp. 23-28, 1996.

31. S. J. Murdoch. Hot or not: revealing hidden services by their clock skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pp. 27-36, November 2006.

32. A. Hintz. Covert channels in TCP and IP Headers, 2003. http://www.defcon.org/images/defcon-10/dc-10-presentations/dc-10-hintz-convert.ppt

33. W. Mazurczyk and K. Szczypiorski. Steganography of VoIP streams. In I. R. Meersman and Z. T., editors, In *Proceedings of the 3rd International*

Symposium on Information Security (IS'08), Monterrey, Mexico, Vol. 5332 of Lecture Notes in Computer Science, pp. 1001-1018. Springer, Berlin, 2008.

34. W. Mazurczyk, M. Smolarczyk, and K. Szczypiorski. Retransmission steganography and its detection. *Soft Computing*, 15(3):505-515, 2011.

35. S. Wendzel. Protocol hopping covert channels. November 2007. http://www.wendzel.de/dr.org/files/Papers/protocolhopping.txt.

36. A. L. Donaldson, J. McHugh, and K. A. Nyberg. Covert channels in trusted LANs. In *Proceedings of the 11th NBS/NCSC National Computer Security Conference*, pp. 226-232, October 1988.

37. S. Wendzel and J. Keller. Design and implementation of an active warden addressing protocol switching covert channels. In *Proceedings of the 7th International Conference on Internet Monitoring and Protection (ICIMP 2012)*, pp. 1-6. IARIA, 2012.

38. S. Zander. *Performance of selected noisy covert channels and their countermeasures in IP networks (tesis doctoral)*. Swinburne University of Technology. May 2010.

39. Belov, Dmitry & Armstrong, Ronald. (2011). Distributions of the Kullback-Leibler divergence with applications. *The British journal of mathematical and statistical psychology*. 64. 291-309. 10.1348/000711010X522227.

40. Zhai, Jiangtao & Liu, Guangjie & Dai, Yuewei. (2010). A Covert Channel Detection Algorithm Based on TCP Markov Model. *Multimedia Information Networking and Security*, International Conference on. 893-897. 10.1109/MINES.2010.190.

41. *Ars generalis ultima*. Base de datos Ramon Llull. Barcelona: Centre de documentació Ramon Llull. Universitat de Barcelona.

42. A. M. TURING, I.—COMPUTING MACHINERY AND INTELLIGENCE, *Mind*, Volume LIX, Issue 236, October 1950, Pages 433–460, https://doi.org/10.1093/mind/LIX.236.433

43. T. Sohn, J. Seo, and J. Moon. A study on the covert channel detection of TCP/IP header using support vector machine. In *Proceedings of 5th International Conference on Information and Communications Security*, pp. 313-324, October 2003.

44. R. Archibald and D. Ghosal. A comparative analysis of detection metrics for covert timing channels. *Computers & Security*, 45:284-292, 2014.

45. Y. Shen, L. Huang, X. Lu, and W. Yang, A novel comprehensive steganalysis of transmission control protocol/Internet protocol covert

channels based on protocol behaviors and support vector Machine. *Security and Communication Networks*, vol. 8, no. 7, pp. 1279-1290, 2015.

46. M. Castellano. (2009). *Modelización estadística con Redes Neuronales. Aplicaciones a la Hidrología, Aerobiología y Modelización de Procesos (tesis doctoral)*. Universidade da Coruña. A Coruña. España.

47. E. Tumoian and M. Anikeev, Network based detection of passive covert channels in TCP/IP. In *Local Computer Networks, 2005. 30th Anniversary*. The IEEE Conference. 2005, pp. 802-809: IEEE.

48. Y. Sun, L. Zhang and C. Zhao, A Study of Network Covert Channel Detection Based on Deep Learning. *2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, Xi'an, 2018, pp. 637-641, doi: 10.1109/IMCEC.2018.8469669.

49. S. Wendzel and S. Zander. Detecting protocol switching covert channels. In *Proceedings of the 37th IEEE Conference on Local Computer Networks (LCN)*, pp. 280-283, 2012.

50. https://github.com/cdpxe/CCEAP/tree/master/documentation, vist Maig 2020.

51. https://www.linphone.org/technical-corner/linphone, vist Maig 2020.

52. https://www.netfilter.org, vist Maig 2020.

53. https://materials.rangeforce.com/tutorial/2020/02/09/Suricata-Intrusion-Prevention-System/, vist Maig 2020.

54. https://docs.mongodb.com/manual/release-notes/3.6/, vist Març 2020.

55. https://api.mongodb.com/python/current/changelog.html, vist Març 2020.

56. https://dpkt.readthedocs.io/en/latest/, vist Març 2020.

57. Travis E. Oliphant. **A guide to NumPy**, USA: Trelgol Publishing, (2006).

58. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011

59. https://docs.python.org/3.6/library/multiprocessing.html, vist Març 2020.

60. https://docs.python.org/3.6/library/threading.html, vist en Maig 2020.

61. S. Zander, G. J. Armitage, and P. Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials 9* (2007), 44–57. Issue 3.

62. Daniel Lerch-Hostalot, David Megías, “LSB matching steganalysis based on patterns of pixel differences and random embedding”, *Computers & Security*, Volume 32, Pages 192-206. February 2013. ISSN: 0167-4048. http://dx.doi.org/10.1016/j.cose.2012.11.005.

63. X. Zhang, C. Liang, Q. Zhang, Y. Li, J. Zheng, Y. Tan. Building covert timing channels by packet rearrangement over mobile networks. *Information Sciences*, Volumes 445–446, 2018, Pages 66-78, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2018.03.007.

64. Y. Liu, D. Ghosal, F. Armknecht, A. Sadeghi, S. Schulz, and S. Katzenbeisser. Robust and Undetectable Steganographic Timing Channels for i.i.d. Traffic. *Information Hiding*, 12th International Conference, IH 2010, LNCS 6387, pp. 193–207, 2010.

65. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html, vist Abril 2020.

66. https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.kl_div.html, vist Abril 2020.

67. G. J. Simmons. The prisoners’ problem and the subliminal channel.In *CRYPTO*, pages 51–67,1983.

68. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, vist Abril 2020.

9. Annexos

ANNEX A – k-test.py

Codi del programari que avalua estadísticament amb els tests KS i KLD dos vectors.

```
import numpy as np
from scipy import special, stats

# Vectors amb els grups de PDU
#cc = []
#ncc = []

minv = 0
maxv = 100

cc = acc[minv:maxv]
ncc = ancc[minv:maxv]

min_len = min([min(cc), min(ncc)])
max_len = max([max(cc), max(ncc)])

icc = list(range(min_len, max_len + 1))
incc = list(range(min_len, max_len+1))

hcc, icc = np.histogram(cc, bins=icc)
hncc, incc = np.histogram(ncc, bins=incc)

chcc = np.cumsum(hcc)
chncc = np.cumsum(hncc)

print('- Distribucions:\n\t- CC: {}\n\t- Not CC: {}'.format(hcc,
hncc))
print('- Cumulative:\n\t- CC: {}\n\t- Not CC: {}'.format(chcc,
chncc))
print('- Intervals:\n\t- icc: {}\n\t- incc: {}'.format(icc, incc))

print('\nKLD')
print(' - Resultat cum: {}'.format(special.kl_div(chncc, chcc)))
print(' - Resultat cum: {}'.format(np.mean(special.kl_div(chncc,
chcc))))

print('\nKS')
print(' - Resultat: {}'.format(stats.ks_2samp(hcc, hncc)))
```

ANNEX B – tfm_cc_rtp.py

Codi font del programari que permet ocultar i obtenir un missatge (vegeu 4.1.3).

```
import sys
import re
import queue
import copy
import fnfqueue

# GLOBAL VARIABLES

# Iptables nfqueue
NFQUEUE = 1

# RTP type of PDU
RTP_AUDIO = 124
RTP_VIDEO = 103
RTP_END = 0


def receiver(verbose, covert_channel):
  # to control the packets
  old_packet = None
  counter_packets = 0
  total_packets = 0

  # to control the length of the message to hide
  secret_message = ''
  first_time = True
  message_finished = False

  # Iptables queue connection
  try:
    queue = 1
    conn = fnfqueue.Connection()
    q = conn.bind(queue)
    q.set_mode(0xffff, fnfqueue.COPY_PACKET)
  except PermissionError:
    print('Are you sure you\'re root or NFQueue is not in use?
Exit..')
    exit()

  if covert_channel:
    while True:
      try:
        for act_packet in conn:
```

```
          act_type = int.from_bytes(act_packet.payload[29:30],
byteorder='big')
          if act_type == RTP_AUDIO or act_type == RTP_VIDEO or not
message_finished:
            total_packets += 1
            if old_packet is None:
              old_packet = copy.copy(act_packet)
            else:
              old_type = int.from_bytes(old_packet.payload[29:30],
byteorder='big')
              old_seq = int.from_bytes(old_packet.payload[30:32],
byteorder='big')

              old_packet.mangle()
              old_packet = copy.copy(act_packet)
              counter_packets += 1
              if verbose == 2:
                print('     * type: {} | seq: {} | counter:
{}'.format(old_type,

old_seq,

counter_packets))

              if act_type != old_type:
                if verbose >= 1:
                  print('- x: {} | type: {} | seq: {} | counter:
{}'.format((counter_packets % 2),

old_type,

old_seq,

counter_packets))
                secret_message += str(counter_packets % 2)
                counter_packets = 0
                if (first_time and len(secret_message) == 7) or (
                  not first_time and len(secret_message) == 8):
                  try:
                    secret_message = int(secret_message, 2)
                    secret_message =
secret_message.to_bytes((secret_message.bit_length() + 7) // 8,

'big').decode()
                    if verbose == 0:
                      sys.stdout.write(str(secret_message))
                      sys.stdout.flush()
                    else:
```

```
print('++++++++++++++++++++++++++++++++++++++++++++++
{}'.format(str(secret_message)))
                    secret_message = ''
                    first_time = False
                  except UnicodeDecodeError:
                    if verbose == 0:
                      sys.stdout.write('-UE-')
                      sys.stdout.flush()
                    else:

print('++++++++++++++++++++++++++++++++++++++++++++++ -UE-')
                    secret_message = ''
                    first_time = False
                  except TypeError:
                    if verbose == 0:
                      sys.stdout.write('-TE-')
                      sys.stdout.flush()
                    else:

print('++++++++++++++++++++++++++++++++++++++++++++++ -TE-')
                    secret_message = ''
                    first_time = False
          elif act_type == 0:
            message_finished = True
          else:
            act_packet.mangle()
      except KeyboardInterrupt:
        conn.close()
        print('bye, bye!')
        exit()
      except:
        pass
  else:
    try:
      for act_packet in conn:
        act_packet.mangle()
    except:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      print('exception: {}: {}'.format(exc_type, exc_tb.tb_lineno))


def sender(verbose, covert_channel, secret_message, diff=0):
  # to control the packets
  old_packet = None
  exc_audios = queue.Queue()
  exc_videos = queue.Queue()
  counter_packets = 0
  type_sending = 0
```

```
  # to control the length of the message to hide
  counter_bits = 0

  # Iptables queue connection
  try:
    conn = fnfqueue.Connection()
    q = conn.bind(NFQUEUE)
    q.set_mode(0xffff, fnfqueue.COPY_PACKET)
  except PermissionError:
    print('Are you sure you\'re root or NFQueue is not in use?
Exit..')
    exit()

  if verbose == 2:
    print('------ SECRET MESSAGE PARITY: {}'.format(secret_message))

  if covert_channel:
    while counter_bits < len(secret_message):
      try:
        for act_packet in conn:
          act_type = int.from_bytes(act_packet.payload[29:30],
byteorder='big')
          if act_type == RTP_AUDIO or act_type == RTP_VIDEO:
            if old_packet is None:
              old_packet = copy.copy(act_packet)
              type_sending =
int.from_bytes(old_packet.payload[29:30], byteorder='big')
            else:
              old_type = int.from_bytes(old_packet.payload[29:30],
byteorder='big')
              old_seq = int.from_bytes(old_packet.payload[30:32],
byteorder='big')

              if act_type == old_type and old_type == type_sending:
                counter_packets += 1
                if verbose == 2:
                  print('    > sending because equals')
                  print('      ---- sending type: {} | seq: {} |
counter: {}'.format(old_type,

old_seq,

counter_packets))
                old_packet.mangle()
                old_packet = copy.copy(act_packet)
              elif act_type != old_type and old_type ==
type_sending:
```

```
                parity = ((counter_packets + 1) % 2 ==
int(secret_message[counter_bits]))
                if parity:
                  old_packet.mangle()
                  old_packet = copy.copy(act_packet)
                  if verbose == 2:
                    print('   > sending old because of parity')
                  if verbose >= 1:
                    print('- x: {} | type: {} | seq: {} | counter:
{}'.format((counter_packets + 1) % 2,

old_type,

old_seq,

(counter_packets + 1)))
                  counter_packets = 0
                  counter_bits += 1
                  type_sending = act_type
                elif counter_packets == 0:
                  if verbose == 2:
                    print('   > sending old because must be sent one
of this type')
                    print('      ---- sending type: {} | seq: {} |
counter: {}'.format(old_type,

old_seq,

(counter_packets + 1)

))
                  old_packet.mangle()
                  old_packet = copy.copy(act_packet)
                  counter_packets += 1
                else:
                  if verbose == 2:
                    print('   * NOT sending old packet because of
parity: {}'.format(type_sending))
                  if verbose >= 1:
                    print('- x: {} | type: {} | seq: {} | counter:
{}'.format(counter_packets % 2,

old_type,

old_seq,

counter_packets))
                  if old_type == RTP_AUDIO:
                    exc_audios.put(old_packet)
```

```
                    else:
                      exc_videos.put(old_packet)
                    old_packet = copy.copy(act_packet)
                    counter_packets = 0
                    counter_bits += 1
                    type_sending = act_type

                  if not exc_audios.empty() and act_type == RTP_AUDIO:
                    if verbose == 2:
                      print('    + sending excedeed audio packets')
                    counter_packets = 0
                    while not exc_audios.empty():
                      exc_packet = exc_audios.get()
                      exc_seq =
int.from_bytes(exc_packet.payload[30:32], byteorder='big')
                      exc_packet.mangle()
                      counter_packets += 1
                      if verbose == 2:
                        print('      ---- sending type: {} | seq: {} |
counter: {}'.format(act_type,

exc_seq,

counter_packets))
                  if not exc_videos.empty() and act_type == RTP_VIDEO:
                    if verbose == 2:
                      print('    + sending excedeed video packets')
                    counter_packets = 0
                    while not exc_videos.empty():
                      exc_packet = exc_videos.get()
                      exc_seq =
int.from_bytes(exc_packet.payload[30:32], byteorder='big')
                      exc_packet.mangle()
                      counter_packets += 1
                      if verbose == 2:
                        print('      ---- sending type: {} | seq: {} |
counter: {}'.format(act_type,

exc_seq,

counter_packets))

                elif old_type != type_sending:
                  if verbose == 2:
                    print('    * NOT sending because not same type that
should be sending: {}'.format(type_sending))
                  if old_type == RTP_AUDIO:
                    exc_audios.put(old_packet)
                  else:
```

```
                    exc_videos.put(old_packet)
                  old_packet = copy.copy(act_packet)
            else:
              act_packet.mangle()
        except KeyboardInterrupt:
          conn.close()
          print('bye, bye!')
          exit()
        except IndexError:
          pass
        except:
          exc_type, exc_obj, exc_tb = sys.exc_info()
          print('exception: {}: {}'.format(exc_type,
exc_tb.tb_lineno))

    old_packet.payload = old_packet.payload[:29] + bytes([RTP_END])
+ old_packet.payload[30:]
    old_packet.mangle()
    print('Missatge enviat')
    try:
      # To continue with the videocall
      for act_packet in conn:
        act_packet.mangle()
    except:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      print('exception: {}: {}'.format(exc_type, exc_tb.tb_lineno))
  else:
    try:
      for act_packet in conn:
        act_packet.mangle()
    except:
      exc_type, exc_obj, exc_tb = sys.exc_info()
      print('exception: {}: {}'.format(exc_type, exc_tb.tb_lineno))


# ---
# argv_error function
# ---
# Print the argument parsing error
def argv_error():
  print('ERROR: arguments:')
  print('./tfm_cc_rtp.py [-v|-d] (decoder|encoder (ncc [% differs
from real traffic]|cc [\'Message\'|-t file_name]))')
  sys.exit()


# ---
# Main function
# ---
```

```
# Based on the arguments, decide if the program should be the
encoder or decoder
if __name__ == '__main__':
  if len(sys.argv) < 2 or len(sys.argv) > 5:
    argv_error()

  diff = 0
  covert_channel = False
  secret_message = ''

  more = 1
  verbose = 0
  if sys.argv[1] == '-v':
    verbose = 1
  elif sys.argv[1] == '-d':
    verbose = 2
  else:
    more = 0

  if sys.argv[2 + more] == 'cc':
    covert_channel = True
  elif sys.argv[2 + more] == 'ncc':
    if len(sys.argv) == (6 + more):
      diff = int(sys.argv[5 + more])
  else:
    argv_error()

  if sys.argv[1 + more] == 'encoder':
    if covert_channel:
      string = sys.argv[3 + more]
      if string == '-t':
        f = open(sys.argv[4 + more], 'r')
        string = f.read()
        f.close()
      secret_message = bin(int.from_bytes(string.encode(), 'big'))
      secret_message = re.sub('^..', '', secret_message)
    sender(verbose, covert_channel, secret_message)
  elif sys.argv[1 + more] == 'decoder':
    receiver(verbose, covert_channel)
  else:
    argv_error()
```

ANNEX C – detector

Codi font del programari que permet detectar un flux RTP que serveix per ocultar un missatge (vegeu 4.2).

detector.py

Funció principal.

```
# File to init the program

# Imports python libraries
import sys
import argparse

# Imports local libraries
from bin.Init import Init


# Main function
def main(argv):
  # Parse the parameters
  parser = argparse.ArgumentParser(description='Network
Steganography Detector based on OpenAI.')
  parser.add_argument('--verbosity',
                      help='1 - ERROR, 2 - WARNING, 3 - INFO, 4 -
INFO MACHINE LEARNING, 10 - DEBUG',
                      default=0, type=int)
  group = parser.add_mutually_exclusive_group(required=True)
  group.add_argument('-d', '--daemon',
                     help='start as a daemon over the configure
interface in the settings file',
                     action='store_true')
  group.add_argument('-p', '--pcapfile',
                     help='analyze a PCAP file specifying the
traffic type [filename [covert|overt]]',
                     nargs='*', default=[None, None], type=str)
  group.add_argument('-t', '--training',
                     help='train the ML algorithm',
action='store_true')
  args = parser.parse_args()

  if args.pcapfile and len(args.pcapfile) > 2:
    print('PARSER ERROR: maximum 2 PCAP parametres: [filename
[covert|overt]]')
    exit()
  elif args.pcapfile[0] and len(args.pcapfile) == 2 and \
```

```
    args.pcapfile[1] != 'covert' and args.pcapfile[1] != 'overt':
    print('PARSER ERROR: Traffic type should be \'covert\' or
\'overt\' or nothing at all')
    exit()
  if (args.verbosity < 0 or args.verbosity > 4) and args.verbosity
!= 10:
    print('PARSER ERROR: verbosity should be 1 - ERROR, 2 - WARNING,
3 - INFO, 4 - INFO MACHINE LEARNING, 10 - DEBUG')
    exit()

  # Call the init function
  main_app = Init(args.daemon, args.training, args.pcapfile,
args.verbosity)
  main_app.Init_startup(args.pcapfile, args.training)
  main_app.Init_main_processing(args.daemon, args.training,
args.pcapfile)


# Init
if __name__ == '__main__':
  main(sys.argv[1:])
```

__init__.py

Gestiona els logs.

```
# Logging

# Imports python libraries
import logging.config
from os import path, remove

# Imports local libraries
from conf import settings as cfg
from .Monitor import Monitor_Data
from .Database import Database
from .Init import Init
from .Monitor import Monitor
from .Network import Network
from .Packets_Queue import Packets_Queue
from .Pcap import Pcap
from .Processor import Processor
from .Machine_Learning import Machine_Learning
```

```
# If applicable, delete the existing log file to generate a fresh
log file during each execution
if path.isfile(cfg.LOGGING_FILE):
  remove(cfg.LOGGING_FILE)

# Create the Logger
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

# Create a Formatter for formatting the log messages
logger_formatter = logging.Formatter(cfg.LOGGING_FORMAT)

# Create the Handler for logging data to a file
logger_fh = logging.FileHandler(cfg.LOGGING_FILE)
logger_fh.setLevel(logging.DEBUG)

# Add the Formatter to the Handler
logger_fh.setFormatter(logger_formatter)

# Create the Handler for logging data to console
logger_sh = logging.StreamHandler()
logger_sh.setLevel(logging.DEBUG)

# Add the Formatter to the Handler
logger_sh.setFormatter(logger_formatter)

# Add the Handlers to the Logger
logger.addHandler(logger_fh)
logger.addHandler(logger_sh)
```

Database.py

API de la base de dades.

```
# Database class

# Imports python libraries
from pymongo import MongoClient, errors
from bson import ObjectId
from datetime import datetime
import sys
import logging

# Import local libraries
from conf import variables as vrb
from conf.settings import DB_PCAP
```

```
# Class to manage the app's db
class Database:

  # Init method
  def __init__(self, log_level, db_server, db_port, db_name,
sync_queue):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.SQ = sync_queue
    self.db_name = db_name

    if self.log_level >= vrb.INFO:
      self.logger.info('Accessing database..')
    try:
      self.client = MongoClient(db_server, db_port)
      self.db = self.client[self.db_name]
      if self.log_level >= vrb.DEBUG:
        self.logger.debug(self.db)
    except errors as err:
      self.logger.error('Error with database: {}'.format(err))
      self.SQ.put('KILL')

    if self.log_level >= vrb.INFO:
      self.logger.info('Database ready!')

  # Insert ICMP packets into the memory database
  def Database_insert_ICMP_packet(self, pkt):
    self.db.ICMP.insert_one(
      {
        'Date': datetime.now().strftime('%d/%m/%Y %H:%M:%S.%f'),
        'Source_IP': pkt[0],
        'Dest_IP': pkt[1],
        'Type': pkt[2],
        'Code': pkt[3],
        'Checksum': pkt[4],
        'Header': pkt[5],
        'Payload': str(pkt[6]),
        'cc': pkt[7]
      }
    )

  # Insert TCP packets into the memory database
  def Database_insert_TCP_packet(self, pkt):
    self.db.TCP.insert_one(
      {
        'Date': datetime.now().strftime('%d/%m/%Y %H:%M:%S.%f'),
        'Source_IP': pkt[0],
```

```
        'Dest_IP': pkt[1],
        'Source_Port': pkt[2],
        'Dest_Port': pkt[3],
        'Sequence_Number': pkt[4],
        'ACK_Number': pkt[5],
        'Flags': pkt[6],
        'Window': pkt[7],
        'Checksum': pkt[8],
        'Urgent_Pointer': pkt[9],
        'Data': pkt[10],
        'cc': pkt[11]
      }
    )

  # Insert UDP packets into the memory database
  def Database_insert_UDP_packet(self, pkt):
    self.db.UDP.insert_one(
      {
        'Date': datetime.now().strftime('%d/%m/%Y %H:%M:%S.%f'),
        'Source_IP': pkt[0],
        'Dest_IP': pkt[1],
        'Source_Port': pkt[2],
        'Dest_Port': pkt[3],
        'Length': pkt[4],
        'Checksum': pkt[5],
        'UDP_type': pkt[6],
        'Data': pkt[7],
        'cc': pkt[8]
      }
    )

  # Reading memory database
  def Database_get_ICMP_packets(self):
    aggregate_query = [{
      '$group': {
        '_id': {
          'Source_IP': '$Source_IP',
          'Dest_IP': '$Dest_IP'
        }
      }
    }]
    packets = []
    flows = list(self.db.ICMP.aggregate(aggregate_query))
    for flow in flows:
      find_query = {
        'Source_IP': flow['_id']['Source_IP'],
        'Dest_IP': flow['_id']['Dest_IP']
      }
      packets.append(list(self.db.ICMP.find(find_query)))
```

```
    return packets

  def Database_get_TCP_packets(self):
    aggregate_query = [{
      '$group': {
        '_id': {
          'Source_IP': '$Source_IP',
          'Source_Port': '$Source_Port',
          'Dest_IP': '$Dest_IP',
          'Dest_Port': '$Dest_Port'
        }
      }
    }]
    packets = []
    flows = list(self.db.TCP.aggregate(aggregate_query))
    for flow in flows:
      find_query = {
        'Source_IP': flow['_id']['Source_IP'],
        'Source_Port': flow['_id']['Source_Port'],
        'Dest_IP': flow['_id']['Dest_IP'],
        'Dest_Port': flow['_id']['Dest_Port']
      }
      packets.append(list(self.db.TCP.find(find_query)))
    return packets

  def Database_get_UDP_flows_id(self, tagged):
    if tagged:
      aggregate_query = [
        {'$match': {'$or': [{'cc': 1}, {'cc': 0}]}},
        {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
      ]
    else:
      aggregate_query = [
        {'$match': {'cc': 2}},
        {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
      ]

    flows = list(self.db.UDP.aggregate(aggregate_query))
    packets = []
    for flow in flows:
      find_query = {
        'Source_IP': flow['_id']['Source_IP'],
        'Dest_IP': flow['_id']['Dest_IP']
      }
      get_id = {'_id': 1, 'cc': 1}
      packets.append(list(self.db.UDP.find(find_query,
get_id).sort('Date').limit(1)))
```

```
    return packets

  def Database_get_TCP_flows_id(self, tagged):
    if tagged:
      aggregate_query = [
        {'$match': {'$or': [{'cc': 1}, {'cc': 0}]}},
        {'$group': {'_id': {'Source_IP': '$Source_IP',
'Source_Port': '$Source_Port',
                            'Dest_IP': '$Dest_IP', 'Dest_Port':
'$Dest_Port'}}}
      ]
    else:
      aggregate_query = [
        {'$match': {'cc': 2}},
        {'$group': {'_id': {'Source_IP': '$Source_IP',
'Source_Port': '$Source_Port',
                            'Dest_IP': '$Dest_IP', 'Dest_Port':
'$Dest_Port'}}}
      ]

    flows = list(self.db.TCP.aggregate(aggregate_query))
    packets = []
    for flow in flows:
      find_query = {
        'Source_IP': flow['_id']['Source_IP'],
        'Source_Port': flow['_id']['Source_Port'],
        'Dest_IP': flow['_id']['Dest_IP'],
        'Dest_Port': flow['_id']['Dest_Port']
      }
      get_id = {'_id': 1, 'cc': 1}
      packets.append(list(self.db.TCP.find(find_query,
get_id).sort('Date').limit(1)))
    return packets

  def Database_get_ICMP_flows_id(self, tagged):
    if tagged:
      aggregate_query = [
        {'$match': {'$or': [{'cc': 1}, {'cc': 0}]}},
        {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
      ]
    else:
      aggregate_query = [
        {'$match': {'cc': 2}},
        {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
      ]

    flows = list(self.db.ICMP.aggregate(aggregate_query))
```

```
    packets = []
    for flow in flows:
      find_query = {
        'Source_IP': flow['_id']['Source_IP'],
        'Dest_IP': flow['_id']['Dest_IP'],
      }
      get_id = {'_id': 1, 'cc': 1}
      packets.append(list(self.db.ICMP.find(find_query,
get_id).sort('Date').limit(1)))
    return packets

  def Database_get_RTP_flows_id(self, tagged):
    if self.db_name == DB_PCAP:
      if tagged:
        aggregate_query = [
          {'$match': {'$or': [{'cc': 1}, {'cc': 0}]}},
          {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
        ]
      else:
        aggregate_query = [
          {'$match': {'cc': 2}},
          {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
        ]
    else:
      if tagged:
        aggregate_query = [
          {'$match': {'$and': [{'$or': [{'UDP_type': 0},
{'UDP_type': 196}]}, {'$or': [{'cc': 1}, {'cc': 0}]}]}},
          {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
        ]
      else:
        aggregate_query = [
          {'$match': {'$and': [{'UDP_type': 0}, {'cc': 2}]}},
          {'$match': {'$and': [{'$or': [{'UDP_type': 0},
{'UDP_type': 196}]}, {'cc': 2}]}},
          {'$group': {'_id': {'Source_IP': '$Source_IP', 'Dest_IP':
'$Dest_IP'}}}
        ]

    flows = list(self.db.UDP.aggregate(aggregate_query))
    packets = []
    for flow in flows:
      find_query = {
        'Source_IP': flow['_id']['Source_IP'],
        'Dest_IP': flow['_id']['Dest_IP'],
      }
```

```
      get_id = {'_id': 1, 'cc': 1}
      packets.append(list(self.db.UDP.find(find_query,
get_id).sort('Date').limit(1)))
    return packets

  def Database_get_RTP_traffic_by_id(self, id):
    IPs = list(self.db.UDP.find({'_id': ObjectId(id)}, {'Source_IP':
1, 'Dest_IP': 1, '_id': 0}))
    return list(self.db.UDP.find(
      {'Source_IP': IPs[0]['Source_IP'], 'Dest_IP':
IPs[0]['Dest_IP']},
      {'Data': 1, '_id': 0}).sort('Date')
                )

  def Database_get_RTP_identification_by_id(self, id):
    IP = self.db.UDP.find({'_id': ObjectId(id)}, {'Source_IP': 1,
'Dest_IP': 1, '_id': 0})
    return [IP[0]['Source_IP'], IP[0]['Dest_IP']]

  def Database_drop_database(self):
    if self.db_name == DB_PCAP:
      self.db.command('dropDatabase')
```

Flow.py

Gestiona les converses a nivell de protocol.

```
# Import python libraries
import logging

# Import local libraries
from bin.Database import Database
from bin.Monitor import Monitor_Data
from conf import variables as vrb


class Flow:

  def __init__(self, log_level, db_server, db_port, db_name,
sync_queue):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.SQ = sync_queue
    self.db_server = db_server
    self.db_port = db_port
```

```
    self.db_name = db_name

  def Flow_fork_database(self):
    self.DB = Database(self.log_level, self.db_server, self.db_port,
self.db_name, self.SQ)

  def Flow_get_flows(self, prot, tagged):
    flows_returned = []
    while True:
      for flow in getattr(self.DB, 'Database_get_' + prot +
'_flows_id')(tagged):
        if flow not in flows_returned:
          flows_returned.append(flow)
          yield flow

  def Flow_ICMP(self, tagged):
    self.Flow_fork_database()
    flows = self.Flow_get_flows('ICMP', tagged)

    if self.log_level >= vrb.INFO:
      self.logger.debug('Flow_ICMP: get packets from database..')

    while True:
      flow = next(flows)

      # flow_id will be the objectID assigned by the database of the
first packet inserted
      # The database will ensure it's the only one
      flow_id = str(flow[0]['_id'])

      # Update the status dict
      if Monitor_Data.Monitor_Data_get_status_flow_ICMP(flow_id) is
None:
        flow_status = int(flow[0]['cc'])
        Monitor_Data.Monitor_Data_update_status_flow_ICMP(flow_id,
flow_status)

      # Update counters
      Monitor_Data.Monitor_Data_update_counter_flow_ICMP(flow_id,
len(flow))
      Monitor_Data.Monitor_Data_update_counter_flows_ICMP()

  def Flow_TCP(self, tagged):
    self.Flow_fork_database()

    if self.log_level >= vrb.INFO:
      self.logger.debug('Flow_TCP: get packets from database..')

    flows = self.Flow_get_flows('TCP', tagged)
```

```
    while True:
      flow = next(flows)

      # flow_id will be the objectID assigned by the database of the
first packet inserted
      # The database will ensure it's the only one
      flow_id = str(flow[0]['_id'])

      # Update the status dict
      if Monitor_Data.Monitor_Data_get_status_flow_TCP(flow_id) is
None:
        flow_status = int(flow[0]['cc'])
        Monitor_Data.Monitor_Data_update_status_flow_TCP(flow_id,
flow_status)

      # Update counters
      Monitor_Data.Monitor_Data_update_counter_flow_TCP(flow_id,
len(flow))
      Monitor_Data.Monitor_Data_update_counter_flows_TCP()

  def Flow_UDP(self, tagged):
    self.Flow_fork_database()

    if self.log_level >= vrb.INFO:
      self.logger.debug('Flow_UDP: get packets from database..')

    flows = self.Flow_get_flows('UDP', tagged)
    while True:
      flow = next(flows)

      # flow_id will be the objectID assigned by the database of the
first packet inserted
      # The database will ensure it's the only one
      flow_id = str(flow[0]['_id'])

      # Update the status dict
      if Monitor_Data.Monitor_Data_get_status_flow_UDP(flow_id) is
None:
        flow_status = int(flow[0]['cc'])
        Monitor_Data.Monitor_Data_update_status_flow_UDP(flow_id,
flow_status)

      # Update counters
      Monitor_Data.Monitor_Data_update_counter_flow_UDP(flow_id,
len(flow))
      Monitor_Data.Monitor_Data_update_counter_flows_UDP()

  def Flow_RTP(self, tagged):
    self.Flow_fork_database()
```

```
    if self.log_level >= vrb.INFO:
      self.logger.debug('Flow_RTP: get packets from database..')

    flows = self.Flow_get_flows('RTP', tagged)
    while True:
      flow = next(flows)

      # flow_id will be the objectID assigned by the database of the
first packet inserted
      # The database will ensure it's the only one
      flow_id = str(flow[0]['_id'])

      if self.log_level >= vrb.DEBUG:
        self.logger.debug('Flow: RTP: flow id {}, category
{}'.format(flow_id, flow[0]['cc']))

      # Update the status dict
      if Monitor_Data.Monitor_Data_get_status_flow_RTP(flow_id) is
None:
        flow_status = int(flow[0]['cc'])
        Monitor_Data.Monitor_Data_update_status_flow_RTP(flow_id,
flow_status)

      # Update counters
      Monitor_Data.Monitor_Data_update_counter_flow_RTP(flow_id,
len(flow))
      Monitor_Data.Monitor_Data_update_counter_flows_RTP()
```

Init.py

Gestiona el flux del programari i els processos.

```
# Main class

# Imports python libraries
import sys
import logging
from multiprocessing import Process, Queue
import time

# Imports local libraries
from bin.Network import Network
from bin.Packets_Queue import Packets_Queue
from bin.Processor import Processor
```

```
from bin.Monitor import Monitor
from bin.Pcap import Pcap
from bin.Flow import Flow
from bin.Machine_Learning import Machine_Learning
from conf import settings as cfg
from conf import variables as vrb


# Main
class Init:

  # Init function
  def __init__(self, daemon, training, pcapfile, log_level):
    # Create the variables
    self.log_level = log_level if log_level >= cfg.LOGGING_LEVEL
else cfg.LOGGING_FILE
    self.logger = logging.getLogger(__name__)

    db_name = cfg.DB_PCAP if len(pcapfile) == 1 else cfg.DB_NAME
    if self.log_level >= vrb.DEBUG:
      self.logger.debug('database name: {}'.format(db_name))

    if daemon and self.log_level >= vrb.INFO:
      self.logger.info('Starting as daemon..')
    elif training and self.log_level >= vrb.INFO:
      self.logger.info('Starting for training AI..')
    elif self.log_level >= vrb.INFO:
      self.logger.info('Preparing for analyze a PCAP file..')

    self.Procs = list()
    self.Sync_Queue = Queue()
    self.Mon_Proc = Monitor(self.log_level, self.Sync_Queue)

    self.ML_Proc = Machine_Learning(self.log_level, self.Sync_Queue,
db_name, daemon, training)

    if not pcapfile[0] or len(pcapfile) == 1 or not pcapfile[1]:
      self.Flows_Proc = Flow(self.log_level, cfg.DB_SERVER,
cfg.DB_PORT, db_name, self.Sync_Queue)

    if not training:
      self.Pkts_Proc = Processor(self.log_level, cfg.DB_SERVER,
cfg.DB_PORT, db_name, self.Sync_Queue,
                                 (1 if pcapfile[0] and len(pcapfile)
== 2 and pcapfile[1] == 'covert' else
                                  0 if pcapfile[0] and len(pcapfile)
== 2 and pcapfile[1] == 'overt' else 2))

    if daemon:
```

```
      self.Net_Proc = Network(self.log_level, cfg.NETWORK_INTERFACE,
self.Sync_Queue)
    elif len(pcapfile) == 1 or pcapfile[0]:
      self.Pcap_Proc = Pcap(self.log_level, self.Sync_Queue,
pcapfile)

  # To manage the global interruptions
  def Init_except_hook(self, exctype, value, traceback):
    if exctype == KeyboardInterrupt:
      if self.log_level >= vrb.INFO:
        self.logger.info("Init: INFO: Shutdown activated")
      for proc in self.Procs:
        proc.terminate()
    else:
      sys.__excepthook__(exctype, value, traceback)

  # Create the environment necessary to work
  def Init_startup(self, pcapfile, training):
    # Create the monitor process
    if self.log_level >= vrb.INFO:
      self.logger.info('Starting up the monitor process...')
    Mon_P = Process(target=self.Mon_Proc.Monitor_process,
args=((True if pcapfile else False),))
    Mon_P.daemon = True
    Mon_P.start()
    self.Procs.append(Mon_P)
    if self.log_level >= vrb.INFO:
      self.logger.info('Monitor started!')

    if not training:
      # Create the processes to get into the database the packets
based in the protocol
      if self.log_level >= vrb.INFO:
        self.logger.info('Starting up the TCP processing
process...')
      for i in range(0, cfg.TCP_NUMBER_PROCESS):
        if self.log_level >= vrb.INFO:
          self.logger.info('Starting {} TCP
process..'.format(str(i)))
        TCP_PP = Process(target=self.Pkts_Proc.Processor_live_TCP,
args=())
        TCP_PP.daemon = True
        TCP_PP.start()
        self.Procs.append(TCP_PP)
      if self.log_level >= vrb.INFO:
        self.logger.info('TCP processing started!')

      if self.log_level >= vrb.INFO:
```

```
        self.logger.info('Starting up the UDP processing
process...')
      for i in range(0, cfg.UDP_NUMBER_PROCESS):
        if self.log_level >= vrb.INFO:
          self.logger.info('Starting {} UDP
process..'.format(str(i)))
        UDP_PP = Process(target=self.Pkts_Proc.Processor_live_UDP,
args=())
        UDP_PP.daemon = True
        UDP_PP.start()
        self.Procs.append(UDP_PP)
      if self.log_level >= vrb.INFO:
        self.logger.info('UDP processing started!')

      if self.log_level >= vrb.INFO:
        self.logger.info('Starting up the ICMP processing
process...')
      for i in range(0, cfg.ICMP_NUMBER_PROCESS):
        if self.log_level >= vrb.INFO:
          self.logger.info('Starting {} ICMP
process..'.format(str(i)))
        ICMP_PP = Process(target=self.Pkts_Proc.Processor_live_ICMP,
args=())
        ICMP_PP.daemon = True
        ICMP_PP.start()
        self.Procs.append(ICMP_PP)
      if self.log_level >= vrb.INFO:
        self.logger.info('ICMP processing started!')

    # Create the Flows process
    if len(pcapfile) != 2 or not pcapfile[0]:
      tagged = True if training else False

      if self.log_level >= vrb.INFO:
        self.logger.info('Starting ICMP Flows process..')
      ICMP_FP = Process(target=self.Flows_Proc.Flow_ICMP,
args=(tagged,))
      ICMP_FP.daemon = True
      ICMP_FP.start()
      self.Procs.append(ICMP_FP)
      if self.log_level >= vrb.INFO:
        self.logger.info('ICMP Flows started!')

      if self.log_level >= vrb.INFO:
        self.logger.info('Starting TCP Flows process..')
      TCP_FP = Process(target=self.Flows_Proc.Flow_TCP,
args=(tagged,))
      TCP_FP.daemon = True
      TCP_FP.start()
```

```
      self.Procs.append(TCP_FP)
      if self.log_level >= vrb.INFO:
        self.logger.info('TCP Flows started!')

      if self.log_level >= vrb.INFO:
        self.logger.info('Starting UDP Flows process..')
      UDP_FP = Process(target=self.Flows_Proc.Flow_UDP,
args=(tagged,))
      UDP_FP.daemon = True
      UDP_FP.start()
      self.Procs.append(UDP_FP)
      if self.log_level >= vrb.INFO:
        self.logger.info('UDP Flows started!')

      if self.log_level >= vrb.INFO:
        self.logger.info('Starting RTP Flows process..')
      RTP_FP = Process(target=self.Flows_Proc.Flow_RTP,
args=(tagged,))
      RTP_FP.daemon = True
      RTP_FP.start()
      self.Procs.append(RTP_FP)
      if self.log_level >= vrb.INFO:
        self.logger.info('RTP Flows started!')

  # Start process that need to be listened to
  def Init_main_processing(self, daemon, training, pcapfile):
    if daemon:
      # Create the network process
      if self.log_level >= vrb.INFO:
        self.logger.info('Starting up the network process...')
      NP = Process(target=self.Net_Proc.Network_rcv, args=())
      NP.daemon = True
      NP.start()
      self.Procs.append(NP)
      if self.log_level >= vrb.INFO:
        self.logger.info('Network started!')
    elif len(pcapfile) == 1 or pcapfile[0]:
      self.Pcap_Proc.Pcap_process()
      if self.log_level >= vrb.INFO:
        self.logger.info('waiting for reading packets..')
      __pcap_finished = False
      while not __pcap_finished:
        msg = self.Sync_Queue.get()
        if msg == 'PCAP_FINISHED':
          __pcap_finished = True

    if training:
      MLP = Process(target=self.ML_Proc.ML_training,
args=(cfg.FEATURES, cfg.T, cfg.S))
```

```
      MLP.daemon = True
      MLP.start()
      self.Procs.append(MLP)
      if self.log_level >= vrb.INFO:
        self.logger.info('ML Training mode started!')
    elif len(pcapfile) == 1 or daemon:
      ML_GF_PP = Process(target=self.ML_Proc.ML_get_RTP_features,
args=(cfg.FEATURES, cfg.T, cfg.S))
      ML_GF_PP.daemon = True
      ML_GF_PP.start()
      self.Procs.append(ML_GF_PP)
      ML_EV_PP = Process(target=self.ML_Proc.ML_evaluate,
args=((True if len(pcapfile) == 1 else False),))
      ML_EV_PP.daemon = True
      ML_EV_PP.start()
      self.Procs.append(ML_EV_PP)
      if self.log_level >= vrb.INFO:
        self.logger.info('ML Analyze mode started!')

    if daemon or ( len(pcapfile) == 2 and pcapfile[1] ):
      if self.log_level >= vrb.INFO:
        self.logger.info('waiting for processing packets..')
      while not Packets_Queue.Packets_Queue_empty():
        time.sleep(2)

    time.sleep(2)
    while self.Sync_Queue.empty():
      if self.Sync_Queue.get() == 'KILL':
        self.logger.info('KILL')
        if self.log_level >= vrb.INFO and not daemon:
          self.logger.info('FINISHED! Killing all process..')
        elif daemon:
          self.logger.error('something was wrong! Killing all
processes..')
        for proc in self.Procs:
          proc.terminate()
          proc.join(timeout=1.0)
        if self.log_level >= vrb.INFO:
          self.logger.info('DONE!')
        self.Mon_Proc.Monitor_print_last_report(True)

    sys.excepthook = self.Init_except_hook
```

Machine_Learning.py

Implementa totes les funcions de l'algorisme de *machine learning*.

```
# Import python libraries
import logging
import numpy as np
import os.path
from joblib import dump, load
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier,
RandomForestClassifier

# Import local libraries
from bin.Database import Database
from bin.Packets_Queue import Packets_Queue
from bin.Monitor import Monitor_Data
from conf import variables as vrb
from conf import settings as cfg


class Machine_Learning:

  def __init__(self, log_level, sync_queue, db_name, daemon,
training):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.SQ = sync_queue
    self.db_server = cfg.DB_SERVER
    self.db_port = cfg.DB_PORT
    self.db_name = db_name
    self.daemon = daemon

    if os.path.exists(cfg.CLF_MCC) and os.path.exists(cfg.CLF):
      self.MCC = load(cfg.CLF_MCC)
      self.CLF = load(cfg.CLF)
      if self.log_level >= vrb.INFO_ML:
        self.logger.info('Classifier loaded.')
    else:
      self.logger.error('Files \'{}\' or \'{}\' do not
exist.'.format(cfg.CLF, cfg.CLF_MCC))
      self.SQ.put('KILL')

  def ML_fork_database(self):
    self.DB = Database(self.log_level, self.db_server, self.db_port,
self.db_name, self.SQ)

  def ML_get_flows(self, prot):
    flows_returned = []
    while True:
      for flow in getattr(Monitor_Data(), 'Monitor_Data_get_flows_'
+ prot)():
```

```
        if flow not in flows_returned:
          flows_returned.append(flow)
          yield flow

  def ML_get_counter_PDU(self, pkts):
    counters_PDU = list()
    last_type = 0
    counter = 0
    for pkt in pkts:
      rtp_type = int.from_bytes(pkt['Data'][1:2], byteorder='big')
      if rtp_type == 124 or rtp_type == 103:
        if rtp_type == last_type:
          counter += 1
        elif counter > 0:
          last_type = rtp_type
          counters_PDU.append(counter)
          counter = 0
        else:
          counter += 1
          last_type = rtp_type
    return counters_PDU

  def ML_get_features(self, dataset, T, S, features):
    diffs = list()
    dset_features = list()
    block = [0, dataset[0], dataset[1]]
    dataset = dataset[2:]
    count_groups = 0

    x = range(-S, S + 1)
    A = np.array(np.meshgrid(x, x, indexing='xy')).T.reshape(-1,
S).tolist()
    A_counter = [0] * len(A)

    for count_PDU in dataset:
      block.pop(0)
      block.append(count_PDU)
      b = 1
      while b < T:
        diff = block[0] - block[b]
        diff = diff > S and S or (diff < -S and -S or diff)
        diffs.append(diff)
        b += 1
      i = A.index(diffs)
      A_counter[i] += 1
      count_groups += 1
      if count_groups == features:
        dset_features.append(A_counter[:])
        A_counter = [0] * len(A)
```

```
        count_groups = 0
      diffs.clear()
    return dset_features

  def ML_RTP_get_training_datasets(self):
    dataset_CC = []
    dataset_NCC = []
    counter_flows = 0
    still_flows = True

    flows = self.ML_get_flows('RTP')
    while still_flows:
      flow = next(flows)
      counter_flows += 1
      pkts = self.DB.Database_get_RTP_traffic_by_id(flow[0])
      if self.log_level >= vrb.INFO_ML:
        self.logger.info('Get Training Dataset: {} ->
{}'.format(flow[0], int(flow[2])))
      if int(flow[2]) == 1:
        dataset_CC.extend(self.ML_get_counter_PDU(pkts))
      else:
        dataset_NCC.extend(self.ML_get_counter_PDU(pkts))

      if self.log_level >= vrb.DEBUG:
        self.logger.debug('counter {} | flows
{}'.format(counter_flows,

Monitor_Data.Monitor_Data_get_total_flows_RTP()))
      if counter_flows ==
Monitor_Data.Monitor_Data_get_total_flows_RTP():
        still_flows = False

    total_len = min(len(dataset_NCC), len(dataset_CC))
    return dataset_NCC[:total_len], dataset_CC[:total_len]

  def ML_get_RTP_features(self, features, T, S):
    counter_flows = 0

    self.ML_fork_database()

    flows = self.ML_get_flows('RTP')
    while True:
      flow = next(flows)
      self.logger.info('Getting features for flow
\'{}\'..'.format(flow[0]))

      counter_flows += 1
      pkts = self.DB.Database_get_RTP_traffic_by_id(flow[0])
      groups_PDU = self.ML_get_counter_PDU(pkts)
```

```
      if len(groups_PDU) < (features + T):
        groups_PDU.clear()
        self.logger.error('Flow \'{}\' does not have enough
traffic.'.format(flow[0]))
        if not self.daemon:
          self.SQ.put('KILL')
      else:
        feat = self.ML_get_features(groups_PDU, T, S, features)

Packets_Queue.Packets_Queue_insert_RTP_Group_Features([flow[0],
feat])
        self.logger.info('Got features for flow
\'{}\''.format(flow[0]))

  def ML_evaluate(self, pcap):
    sum_prediction = 0

    self.ML_fork_database()

    while True:
      if self.log_level >= vrb.INFO:
        self.logger.info('Evaluate: getting data set...')
      flow_info = Packets_Queue.Packets_Queue_get_RTP_PDU_Groups()

      self.logger.info('Evaluate: predicting for id
\'{}\'..'.format(flow_info[0]))
      for feat in flow_info[1]:
        sum_prediction +=
int(self.CLF.predict(np.array(feat).reshape(1, -1)))

      if sum_prediction == 0 and self.log_level == vrb.DEBUG:
        self.logger.info('Flow \'{}\' does not seem to have a
network covert channel... by now'.format(flow_info[0]))
      else:
        IP =
self.DB.Database_get_RTP_identification_by_id(flow_info[0])
        self.logger.critical(
          'NETWORK COVERT CHANNEL! With {} MCC metric, Source IP {}
and Destination IP {}, RTP'.format(
            self.MCC, IP[0], IP[1]
          ))
        sum_prediction = 0

      if pcap:
        if self.log_level >= vrb.INFO_ML:
          self.logger.info('Dropping temporal database..')
        self.DB.Database_drop_database()
        self.SQ.put('KILL')
```

```
  def ML_training(self, features, T, S):
    self.ML_fork_database()

    self.logger.info('Training: getting data sets..')
    dataset_NCC, dataset_CC = self.ML_RTP_get_training_datasets()

    self.logger.info('Training: getting features..')
    feat_NCC = self.ML_get_features(dataset_NCC, T, S, features)
    feat_CC = self.ML_get_features(dataset_CC, T, S, features)

    sum_feat = len(feat_NCC)
    X_train, X_test, y_train, y_test =
train_test_split(np.concatenate([feat_NCC, feat_CC]),

np.concatenate([np.zeros(sum_feat), np.ones(sum_feat)]),

test_size=0.2, shuffle=True)

    for i in range(0, cfg.TRAINING_ITERATIONS):
      self.logger.info('Init iteration number {}'.format(i))

      clf =
AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=1
0),
                                 n_estimators=10000,
                                 learning_rate=0.1)
      clf.fit(X_train, y_train)
      predicted = clf.predict(X_test)
      mcc = metrics.matthews_corrcoef(y_test, predicted)
      if mcc > self.MCC:
        self.logger.info('Classifier improved! New MCC metric value:
{}'.format(mcc))
        dump(clf, cfg.CLF)
        dump(mcc, cfg.CLF_MCC)
        self.logger.info('New classifier saved.')

    self.SQ.put('KILL')
```

Monitor.py

Classes *Monitor* i *Monitor_Data* per gestionar el flux del programari.

```
# Monitor class

# Imports python libraries
import time
```

```
import logging
from multiprocessing import Manager

# Import local libraries
from conf import variables as vrb


# Class to monitor the app
class Monitor:

  # init method
  def __init__(self, log_level, sync_queue):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.SQ = sync_queue
    self.__status_matrix = [['FLOW', ['UPDATING', 'STANDBY',
'FINISHED', 'ARCHIVED']],
                            ['AI', ['NOT_STARTED',
                                    'PT1_WORKING', 'PT1_WAITING',
'PT1_FINISHED',
                                    'PS12_WORKING', 'PS12_WAITING',
'PS12_FINISHED',
                                    ]],
                            ['RESULT', ['POSITIVE', 'NEGATIVE',
'SUSPECT']]]

  # Return the string that identifies a status
  # TODO: it's manually done, should be read from 'variables' file
  def __Monitor_get_name_status(self, type_status, status):
    return str(self.__status_matrix[type_status][0] + ': ' +
self.__status_matrix[type_status][1][status])

  # monitor process
  def Monitor_process(self, pcap=False):
    if self.log_level < vrb.INFO:
      return 0

    while True:
      TCP_received_packets =
Monitor_Data.Monitor_Data_get_total_received_TCP()
      UDP_received_packets =
Monitor_Data.Monitor_Data_get_total_received_UDP()
      ICMP_received_packets =
Monitor_Data.Monitor_Data_get_total_received_ICMP()
      ERROR_received_packets =
Monitor_Data.Monitor_Data_get_total_received_ERROR()
      TCP_database_packets =
Monitor_Data.Monitor_Data_get_total_database_TCP()
```

```
      UDP_database_packets =
Monitor_Data.Monitor_Data_get_total_database_UDP()
      ICMP_database_packets =
Monitor_Data.Monitor_Data_get_total_database_ICMP()
      ERROR_database_packets =
Monitor_Data.Monitor_Data_get_total_database_ERROR()
      TCP_flows = Monitor_Data.Monitor_Data_get_flows_TCP()
      UDP_flows = Monitor_Data.Monitor_Data_get_flows_UDP()
      ICMP_flows = Monitor_Data.Monitor_Data_get_flows_ICMP()
      TCP_counter_flows =
Monitor_Data.Monitor_Data_get_total_flows_TCP()
      UDP_counter_flows =
Monitor_Data.Monitor_Data_get_total_flows_UDP()
      ICMP_counter_flows =
Monitor_Data.Monitor_Data_get_total_flows_ICMP()
      RTP_counter_flows =
Monitor_Data.Monitor_Data_get_total_flows_RTP()

      pkts_origin = 'Received' if not pcap else 'Read'

      self.logger.info('------------------------ PACKET LIVE REPORT
-------------------------')
      self.logger.info('-- {}:'.format(pkts_origin))
      self.logger.info('   TCP|UDP|RTP|ICMP|ERROR|TOTAL:
{}|{}|{}|{}|{}|{}'.
                       format(TCP_received_packets,
UDP_received_packets, ICMP_received_packets,
                              ERROR_received_packets,
                              TCP_received_packets +
UDP_received_packets + ICMP_received_packets +
ERROR_received_packets))
      self.logger.info('-- Inserted into the database:')
      self.logger.info('   TCP|UDP|RTP|ICMP|ERROR|TOTAL:
{}|{}|{}|{}|{}|{}'.
                       format(TCP_database_packets,
UDP_database_packets, ICMP_database_packets,
                              ERROR_database_packets,
                              TCP_database_packets +
UDP_database_packets + ICMP_database_packets +
ERROR_database_packets))
      self.logger.info('-- Conversations identified:')
      self.logger.info('   TCP|UDP|RTP|ICMP|TOTAL: {}|{}|{}|{}|{}'.
                       format(TCP_counter_flows, UDP_counter_flows,
RTP_counter_flows, ICMP_counter_flows,
                              TCP_counter_flows + UDP_counter_flows
+ ICMP_counter_flows + RTP_counter_flows))
      self.logger.info('')

      """
```

```
        if TCP_received_packets > TCP_database_packets:
            self.Pipe.send(['TCP', 1])
        if UDP_received_packets > UDP_database_packets:
            self.Pipe.send(['UDP', 1])
        if ICMP_received_packets > ICMP_database_packets:
            self.Pipe.send(['ICMP', 1])
        """
        time.sleep(30)

    def Monitor_print_last_report(self, pcap=False):
        TCP_received_packets =
Monitor_Data.Monitor_Data_get_total_received_TCP()
        UDP_received_packets =
Monitor_Data.Monitor_Data_get_total_received_UDP()
        ICMP_received_packets =
Monitor_Data.Monitor_Data_get_total_received_ICMP()
        ERROR_received_packets =
Monitor_Data.Monitor_Data_get_total_received_ERROR()
        TCP_database_packets =
Monitor_Data.Monitor_Data_get_total_database_TCP()
        UDP_database_packets =
Monitor_Data.Monitor_Data_get_total_database_UDP()
        ICMP_database_packets =
Monitor_Data.Monitor_Data_get_total_database_ICMP()
        ERROR_database_packets =
Monitor_Data.Monitor_Data_get_total_database_ERROR()

        pkts_origin = 'Received' if not pcap else 'Read'

        self.logger.info('------------------------ PACKET SESSION REPORT
-------------------------')
        self.logger.info('-- {}:'.format(pkts_origin))
        self.logger.info('   TCP|UDP|ICMP|ERROR|TOTAL: {}|{}|{}|{}|{}'.
                         format(TCP_received_packets,
UDP_received_packets, ICMP_received_packets,
                                ERROR_received_packets,
                                TCP_received_packets +
UDP_received_packets + ICMP_received_packets +
ERROR_received_packets))
        self.logger.info('-- Inserted into the database:')
        self.logger.info('   TCP|UDP|ICMP|ERROR|TOTAL: {}|{}|{}|{}|{}'.
                         format(TCP_database_packets,
UDP_database_packets, ICMP_database_packets,
                                ERROR_database_packets,
                                TCP_database_packets +
UDP_database_packets + ICMP_database_packets +
ERROR_database_packets))
```

```
class Monitor_Data:
  __manager = Manager()
  __lock = __manager.Lock()

  Counter_Received_Total_ICMP = __manager.Value('i', 0)
  Counter_Received_Total_TCP = __manager.Value('i', 0)
  Counter_Received_Total_UDP = __manager.Value('i', 0)
  Counter_Received_Total_ERROR = __manager.Value('i', 0)

  Counter_Database_Total_ICMP = __manager.Value('i', 0)
  Counter_Database_Total_TCP = __manager.Value('i', 0)
  Counter_Database_Total_UDP = __manager.Value('i', 0)
  Counter_Database_Total_ERROR = __manager.Value('i', 0)

  Counter_Flows_ICMP = __manager.dict()
  Counter_Flows_TCP = __manager.dict()
  Counter_Flows_UDP = __manager.dict()
  Counter_Flows_RTP = __manager.dict()
  Status_Flows_ICMP = __manager.dict()
  Status_Flows_TCP = __manager.dict()
  Status_Flows_UDP = __manager.dict()
  Status_Flows_RTP = __manager.dict()

  Counter_Total_Flows_ICMP = __manager.Value('i', 0)
  Counter_Total_Flows_TCP = __manager.Value('i', 0)
  Counter_Total_Flows_UDP = __manager.Value('i', 0)
  Counter_Total_Flows_RTP = __manager.Value('i', 0)

  # Increment total received
  @staticmethod
  def Monitor_Data_increment_total_received_ICMP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Received_Total_ICMP.value += 1

  @staticmethod
  def Monitor_Data_increment_total_received_TCP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Received_Total_TCP.value += 1

  @staticmethod
  def Monitor_Data_increment_total_received_UDP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Received_Total_UDP.value += 1

  @staticmethod
  def Monitor_Data_increment_total_received_ERROR():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Received_Total_ERROR.value += 1
```

```
  # Increment total packets inserted in the database
  @staticmethod
  def Monitor_Data_increment_total_database_ICMP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Database_Total_ICMP.value += 1

  @staticmethod
  def Monitor_Data_increment_total_database_TCP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Database_Total_TCP.value += 1

  @staticmethod
  def Monitor_Data_increment_total_database_UDP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Database_Total_UDP.value += 1

  @staticmethod
  def Monitor_Data_increment_total_database_ERROR():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Database_Total_ERROR.value += 1

  # Get total received
  @staticmethod
  def Monitor_Data_get_total_received_ICMP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Received_Total_ICMP.value

  @staticmethod
  def Monitor_Data_get_total_received_TCP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Received_Total_TCP.value

  @staticmethod
  def Monitor_Data_get_total_received_UDP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Received_Total_UDP.value

  @staticmethod
  def Monitor_Data_get_total_received_ERROR():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Received_Total_ERROR.value

  # Get total database
  @staticmethod
  def Monitor_Data_get_total_database_ICMP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Database_Total_ICMP.value

  @staticmethod
```

```
  def Monitor_Data_get_total_database_TCP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Database_Total_TCP.value

  @staticmethod
  def Monitor_Data_get_total_database_UDP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Database_Total_UDP.value

  @staticmethod
  def Monitor_Data_get_total_database_ERROR():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Database_Total_ERROR.value

  # Increment or add flow
  @staticmethod
  def Monitor_Data_update_counter_flow_ICMP(flow, counter):
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Flows_ICMP[flow] = counter

  @staticmethod
  def Monitor_Data_update_counter_flow_TCP(flow, counter):
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Flows_TCP[flow] = counter

  @staticmethod
  def Monitor_Data_update_counter_flow_UDP(flow, counter):
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Flows_UDP[flow] = counter

  @staticmethod
  def Monitor_Data_update_counter_flow_RTP(flow, counter):
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Flows_RTP[flow] = counter

  # Increment number of total flows
  @staticmethod
  def Monitor_Data_update_counter_flows_ICMP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Total_Flows_ICMP.value += 1

  @staticmethod
  def Monitor_Data_update_counter_flows_TCP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Total_Flows_TCP.value += 1

  @staticmethod
  def Monitor_Data_update_counter_flows_UDP():
    with Monitor_Data.__lock:
```

```
    Monitor_Data.Counter_Total_Flows_UDP.value += 1

  @staticmethod
  def Monitor_Data_update_counter_flows_RTP():
    with Monitor_Data.__lock:
      Monitor_Data.Counter_Total_Flows_RTP.value += 1

  # Get flows
  @staticmethod
  def Monitor_Data_get_flows_ICMP():
    with Monitor_Data.__lock:
      flows = []
      for flow in dict(Monitor_Data.Counter_Flows_ICMP):
        flows.append([flow, Monitor_Data.Counter_Flows_ICMP[flow],
                      Monitor_Data.Status_Flows_ICMP[flow]])
      return flows

  @staticmethod
  def Monitor_Data_get_flows_TCP():
    with Monitor_Data.__lock:
      flows = []
      for flow in dict(Monitor_Data.Counter_Flows_TCP):
        flows.append([flow, Monitor_Data.Counter_Flows_TCP[flow],
                      Monitor_Data.Status_Flows_TCP[flow]])
      return flows

  @staticmethod
  def Monitor_Data_get_flows_UDP():
    with Monitor_Data.__lock:
      flows = []
      for flow in dict(Monitor_Data.Counter_Flows_UDP):
        flows.append([flow, Monitor_Data.Counter_Flows_UDP[flow],
                      Monitor_Data.Status_Flows_UDP[flow]])
    return flows

  @staticmethod
  def Monitor_Data_get_flows_RTP():
    with Monitor_Data.__lock:
      flows = []
      for flow in dict(Monitor_Data.Counter_Flows_RTP):
        flows.append([flow, Monitor_Data.Counter_Flows_RTP[flow],
                      Monitor_Data.Status_Flows_RTP[flow]])
    return flows

  # Get a flow's counter
  @staticmethod
  def Monitor_Data_get_counter_flow_ICMP(flow):
    with Monitor_Data.__lock:
      try:
```

```
        return Monitor_Data.Counter_Flows_ICMP[flow]
      except KeyError as ke:
        return None

  @staticmethod
  def Monitor_Data_get_counter_flow_TCP(flow):
    with Monitor_Data.__lock:
      try:
        return Monitor_Data.Counter_Flows_TCP[flow]
      except KeyError as ke:
        return None

  @staticmethod
  def Monitor_Data_get_counter_flow_UDP(flow):
    with Monitor_Data.__lock:
      try:
        return Monitor_Data.Counter_Flows_UDP[flow]
      except KeyError as ke:
        return None

  @staticmethod
  def Monitor_Data_get_counter_flow_RTP(flow):
    with Monitor_Data.__lock:
      try:
        return Monitor_Data.Counter_Flows_RTP[flow]
      except KeyError as ke:
        return None

  # Get a flow's status
  @staticmethod
  def Monitor_Data_get_status_flow_ICMP(flow):
    with Monitor_Data.__lock:
      try:
        return Monitor_Data.Status_Flows_ICMP[flow]
      except KeyError as ke:
        return None

  @staticmethod
  def Monitor_Data_get_status_flow_TCP(flow):
    with Monitor_Data.__lock:
      try:
        return Monitor_Data.Status_Flows_TCP[flow]
      except KeyError as ke:
        return None

  @staticmethod
  def Monitor_Data_get_status_flow_UDP(flow):
    with Monitor_Data.__lock:
      try:
```

```
            return Monitor_Data.Status_Flows_UDP[flow]
        except KeyError as ke:
            return None

    @staticmethod
    def Monitor_Data_get_status_flow_RTP(flow):
        with Monitor_Data.__lock:
            try:
                return Monitor_Data.Status_Flows_RTP[flow]
            except KeyError as ke:
                return None

    # Update or create a flow's status
    @staticmethod
    def Monitor_Data_update_status_flow_ICMP(flow, status):
        with Monitor_Data.__lock:
            Monitor_Data.Status_Flows_ICMP[flow] = status

    @staticmethod
    def Monitor_Data_update_status_flow_TCP(flow, status):
        with Monitor_Data.__lock:
            Monitor_Data.Status_Flows_TCP[flow] = status

    @staticmethod
    def Monitor_Data_update_status_flow_UDP(flow, status):
        with Monitor_Data.__lock:
            Monitor_Data.Status_Flows_UDP[flow] = status

    @staticmethod
    def Monitor_Data_update_status_flow_RTP(flow, status):
        with Monitor_Data.__lock:
            Monitor_Data.Status_Flows_RTP[flow] = status

    # Get total flows
    @staticmethod
    def Monitor_Data_get_total_flows_ICMP():
        with Monitor_Data.__lock:
            return Monitor_Data.Counter_Total_Flows_ICMP.value

    @staticmethod
    def Monitor_Data_get_total_flows_TCP():
        with Monitor_Data.__lock:
            return Monitor_Data.Counter_Total_Flows_TCP.value

    @staticmethod
    def Monitor_Data_get_total_flows_UDP():
        with Monitor_Data.__lock:
            return Monitor_Data.Counter_Total_Flows_UDP.value
```

```
  @staticmethod
  def Monitor_Data_get_total_flows_RTP():
    with Monitor_Data.__lock:
      return Monitor_Data.Counter_Total_Flows_RTP.value
```

Network.py

API d'accés a la interfície de xarxa.

```
# Network class

# Imports python libraries
import platform
import socket
import logging

# Import local libraries
from bin.Monitor import Monitor_Data
from bin.Packets_Queue import Packets_Queue
from conf import variables as vrb


class Network:

  # Init method
  def __init__(self, log_level, interface, sync_queue):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.iface = interface
    self.SQ = sync_queue

    self.logger.info('Opening socket..')
    try:
      if platform.system() == 'Linux':
        self.sock = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.htons(0x0003))
        self.sock.bind((self.iface, 0))
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 2
** 30)
      else:
        self.logger.error('NotImplementedError: platform not
implemented yet')
        self.SQ.put('KILL')
    except PermissionError:
      self.logger.error('PermissionError: you must be root')
      self.SQ.put('KILL')
    except OSError:
```

```
      self.logger.error('OSError: sure the interface {}
exists?'.format(self.iface))
      self.SQ.put('KILL')
    self.logger.info('Socket open!')

  # Listen to the network
  def Network_rcv(self):
    self.logger.info('Ready to receive packets!')
    while True:
      try:
        packet = self.sock.recv(65565)
        prot = int.from_bytes(packet[23:24], byteorder='big')

        if prot == 6:
          Packets_Queue.Packets_Queue_insert_TCP(packet)
          Monitor_Data.Monitor_Data_increment_total_received_TCP()
        elif prot == 17:
          Packets_Queue.Packets_Queue_insert_UDP(packet)
          Monitor_Data.Monitor_Data_increment_total_received_UDP()
        elif prot == 2:
          Packets_Queue.Packets_Queue_insert_ICMP(packet)
          Monitor_Data.Monitor_Data_increment_total_received_ICMP()
        else:
          Monitor_Data.Monitor_Data_increment_total_received_ERROR()
      except KeyboardInterrupt:
        if self.log_level >= vrb.INFO:
          self.logger.info('Closing sockets...')
        return
```

Packets_Queue.py

Implementa la funcionalitat de la cua de paquets en memòria RAM.

```
# Class to add the packet to the processing queue

# Imports python libraries
from multiprocessing import Manager


# Class to process protocols-based queues
class Packets_Queue:
  __manager = Manager()

  TCP_Queue = __manager.Queue()
  UDP_Queue = __manager.Queue()
  ICMP_Queue = __manager.Queue()
```

```
  RTP_PDU_Groups_Queue = __manager.Queue()

  @staticmethod
  def Packets_Queue_insert_TCP(packet):
    Packets_Queue.TCP_Queue.put(packet)

  @staticmethod
  def Packets_Queue_insert_UDP(packet):
    Packets_Queue.UDP_Queue.put(packet)

  @staticmethod
  def Packets_Queue_insert_ICMP(packet):
    Packets_Queue.ICMP_Queue.put(packet)

  @staticmethod
  def Packets_Queue_insert_RTP_Group_Features(group):
    Packets_Queue.RTP_PDU_Groups_Queue.put(group)

  @staticmethod
  def Packets_Queue_get_TCP():
    return Packets_Queue.TCP_Queue.get()

  @staticmethod
  def Packets_Queue_get_UDP():
    return Packets_Queue.UDP_Queue.get()

  @staticmethod
  def Packets_Queue_get_ICMP():
    return Packets_Queue.ICMP_Queue.get()

  @staticmethod
  def Packets_Queue_get_RTP_PDU_Groups():
    return Packets_Queue.RTP_PDU_Groups_Queue.get()

  @staticmethod
  def Packets_Queue_empty():
    return True if Packets_Queue.TCP_Queue.empty() and
Packets_Queue.UDP_Queue.empty() and Packets_Queue.ICMP_Queue.empty()
else False
```

Pcap.py

API per llegir un fitxer PCAP.

```
# Pcap reading class
```

```
# Imports python libraries
import dpkt
import logging

# Import local libraries
from bin.Monitor import Monitor_Data
from bin.Packets_Queue import Packets_Queue
from conf import variables as vrb


# Class to read pcaps file
class Pcap:

  # Init method
  def __init__(self, log_level, sync_queue, pcapfile):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.SQ = sync_queue
    self.set_analyze = True if len(pcapfile) == 1 else False

    if self.set_analyze:
      self.logger.info('Reading pcap file\'{}\' to analyze
it'.format(pcapfile[0]))
    else:
      self.logger.info('Reading pcap file \'{}\' as \'{}\'
traffic'.format(pcapfile[0], pcapfile[1]))
    try:
      self.file = dpkt.pcap.Reader(open(pcapfile[0], 'rb'))
    except ValueError as e:
      self.logger.error('Error reading the pcap file {}:
{}'.format(pcapfile[0], str(e)))
      self.SQ.put('KILL')

    if self.log_level >= vrb.INFO:
      self.logger.info('PCAP file parsed!')

  # Process packets
  def Pcap_process(self):
    self.logger.info('Getting packets from file...')
    for ts, pkt in self.file:
      prot = int.from_bytes(pkt[23:24], byteorder='big')

      if prot == 6:
        Packets_Queue.Packets_Queue_insert_TCP(pkt)
        Monitor_Data.Monitor_Data_increment_total_received_TCP()
      elif prot == 17:
        Packets_Queue.Packets_Queue_insert_UDP(pkt)
        Monitor_Data.Monitor_Data_increment_total_received_UDP()
      elif prot == 2:
```

```
        Packets_Queue.Packets_Queue_insert_ICMP(pkt)
        Monitor_Data.Monitor_Data_increment_total_received_ICMP()
      else:
        Monitor_Data.Monitor_Data_increment_total_received_ERROR()

    self.logger.info('All packets readed!')
    self.SQ.put('PCAP_FINISHED')
    if not self.set_analyze:
      self.SQ.put('KILL')
```

Processor.py

Inserta els paquets a la base de dades des de la memòria RAM.

```
# Class to process the packets

# Import python libraries
import socket
import logging

# Import local libraries
from bin.Database import Database
from bin.Monitor import Monitor_Data
from bin.Packets_Queue import Packets_Queue
from conf import variables as vrb


# Class to make the first packets classification based in the
protocol
class Processor:

  def __init__(self, log_level, db_server, db_port, db_name,
sync_queue, pcap_type=None):
    self.log_level = log_level
    self.logger = logging.getLogger(__name__)
    self.SQ = sync_queue
    self.db_server = db_server
    self.db_port = db_port
    self.db_name = db_name
    self.pcap_type = pcap_type

  def Processor_fork_database(self):
    self.DB = Database(self.log_level, self.db_server, self.db_port,
self.db_name, self.SQ)
```

```
  # Process Live packets
  def Processor_live_ICMP(self):
    self.Processor_fork_database()
    while True:
      try:
        packet = Packets_Queue.Packets_Queue_get_UDP()

        Source_IP = socket.inet_ntoa(packet[26:30])
        Dest_IP = socket.inet_ntoa(packet[30:34])
        Type = int.from_bytes(packet[34:35], byteorder='big')
        Code = int.from_bytes(packet[35:36], byteorder='big')
        Checksum = int.from_bytes(packet[36:38], byteorder='big')
        Header = int.from_bytes(packet[38:42], byteorder='big')
        Payload = int.from_bytes(packet[42:50], byteorder='big')
        Traffic_CC = self.pcap_type

        self.DB.Database_insert_ICMP_packet([Source_IP, Dest_IP,
Type, Code, Checksum, Header, Payload,
                                             Traffic_CC])

        Monitor_Data.Monitor_Data_increment_total_database_ICMP()
      except KeyboardInterrupt:
        if self.log_level >= vrb.INFO:
          self.logger.info('Closing ICMP process..')
        return

  def Processor_live_TCP(self):
    self.Processor_fork_database()
    while True:
      try:
        packet = Packets_Queue.Packets_Queue_get_TCP()

        Source_IP = socket.inet_ntoa(packet[26:30])
        Dest_IP = socket.inet_ntoa(packet[30:34])
        Source_Port = int.from_bytes(packet[34:36], byteorder='big')
        Dest_Port = int.from_bytes(packet[36:38], byteorder='big')
        Sequence_Number = int.from_bytes(packet[38:42],
byteorder='big')
        ACK_Number = int.from_bytes(packet[42:46], byteorder='big')
        Flags = int.from_bytes(packet[46:48], byteorder='big')
        Window = int.from_bytes(packet[48:50], byteorder='big')
        Checksum = int.from_bytes(packet[50:52], byteorder='big')
        Urgent_Pointer = int.from_bytes(packet[52:54],
byteorder='big')
        Data = packet[54:]
        Traffic_CC = self.pcap_type

        self.DB.Database_insert_TCP_packet([Source_IP, Dest_IP,
Source_Port, Dest_Port, Sequence_Number,
```

```
                                                ACK_Number, Flags,
Window, Checksum, Urgent_Pointer, Data,
                                                Traffic_CC])

        Monitor_Data.Monitor_Data_increment_total_database_TCP()
      except KeyboardInterrupt:
        if self.log_level >= vrb.INFO:
          self.logger.info('Closing TCP process..')
        return

  def Processor_live_UDP(self):
    self.Processor_fork_database()
    while True:
      try:
        packet = Packets_Queue.Packets_Queue_get_UDP()

        Source_IP = socket.inet_ntoa(packet[26:30])
        Dest_IP = socket.inet_ntoa(packet[30:34])
        Source_Port = int.from_bytes(packet[34:36], byteorder='big')
        Dest_Port = int.from_bytes(packet[36:38], byteorder='big')
        Length = int.from_bytes(packet[38:40], byteorder='big')
        Checksum = int.from_bytes(packet[40:42], byteorder='big')
        UDP_type = int.from_bytes(packet[42:43], byteorder='big')
        Data = packet[42:]
        Traffic_CC = self.pcap_type

        self.DB.Database_insert_UDP_packet([Source_IP, Dest_IP,
Source_Port, Dest_Port,
                                            Length, Checksum,
UDP_type, Data, Traffic_CC])

        Monitor_Data.Monitor_Data_increment_total_database_UDP()
      except KeyboardInterrupt:
        if self.log_level >= vrb.INFO:
          self.logger.info('Closing UDP process..')
        return
```

settings.py

Fitxer d'opcions de configuració.

```
# Config file

# Import libraries
import os
```

```
BASE_DIR = os.path.dirname('.')

# DATABASE
DB_SERVER = '127.0.0.1'
DB_PORT = 27017
DB_NAME = 'detector_db'
DB_PCAP = 'detector_pcap_db'

# NETWORK
NETWORK_INTERFACE = 'enp7s0'

# QUEUE
TCP_NUMBER_PROCESS = 10
UDP_NUMBER_PROCESS = 5
ICMP_NUMBER_PROCESS = 1

# LOGGING
LOGGING_LEVEL = 0
LOGGING_FORMAT = '%(asctime)s - %(name)s - %(levelname)s -
%(message)s'
LOGGING_FILE = BASE_DIR + 'logging/detector.log'

# MACHINE LEARNING
FEATURES = 100
T = 3
S = 2
CLF = BASE_DIR + 'dump/classifier.dump'
CLF_MCC = BASE_DIR + 'dump/clf_mcc'
TRAINING_ITERATIONS = 20
```

variables.py

Variables que no s'haurien de modificar.

```
# File to define the global variables

# FLOW STATUS
FLOW_UPDATING = 0
FLOW_STANDBY = 1
FLOW_FINISHED = 2
FLOW_ARCHIVED = 3

# FLOW AI PROCESSING STATUS
FLOW_AI_TRAINING = 0
FLOW_AI_FINISHED = 1
```

```
FLOW_AI_NOT_STARTED = 2
FLOW_AI_PT1_WORKING = 3
FLOW_AI_PT1_WAITING = 4
FLOW_AI_PT1_FINISHED = 5
FLOW_AI_PS12_WORKING = 6
FLOW_AI_PS12_WAITING = 7
FLOW_AI_PS12_FINISHED = 8

# FLOW AI RESULT
FLOW_AI_POSITIVE = 0
FLOW_AI_NEGATIVE = 1
FLOW_AI_SUSPECT = 2
FLOW_AI_TRAINED = 3

# TYPE COVERT CHANNEL
CC_PT1 = 0
CC_PT2 = 1
CC_PT11 = 11

# LOG LEVEL
NONE = 0
ERROR = 1
WARNING = 2
INFO = 3
INFO_ML = 4
DEBUG = 10
```

requeriments.txt

Llista de llibreries que cal instal·lar.

```
pymongo==3.9.0
dpkt==1.9.2
numpy==1.17
sklearn==0.23.0
```