Intelligence Artificielle En Réseaux

- Dispensé par Dr. Msc. Ir. **MWAMBA KASONGO Dahouda**
- Docteur en génie logiciel et systèmes d'information
- Machine and Deep Learning Engineer

- Assisté Ass. Grace PWELE

- E-mail : dahouda37@gmail.com

Heure : 08H00 – 12H00

CHAPITRE 5 Analyse du trafic réseau pour l'apprentissage automatique (ML)

1. Introduction
2. Pourquoi utiliser Wireshark
3. Comment Wireshark fonctionne-t-il?
4. Dans quels cas professionnels utilise-t-on Wireshark ?
5. Pour quel métier utilise-t-on Wireshark ?
6. Lancement de Wireshark
7. Capture des trames sur le réseau

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

1. Introduction a wireshark

❑ Wireshark est un analyseur de protocoles réseau open source.
Il vous permet de capturer et d'analyser le trafic réseau en temps réel.
Que ce soit sur votre propre réseau local ou sur des réseaux distants, Wireshark vous donne une visibilité totale sur les paquets de données qui transitent à travers les câbles et les ondes.

Wireshark est un logiciel d'analyse réseau (sniffer) qui permettant de visualiser l'ensemble des données transitant sur la machine qui l'exécute, et d'obtenir des informations sur les protocoles applicatifs utilisés.
Les octets sont capturés en utilisant la librairie réseau PCAP, puis regroupés en blocs d'informations et analysés par le logiciel.

La dernière version de Wireshark est disponible en téléchargement sur www.wireshark.org.
De nombreuses distributions linux incluent Wireshark dans leur gestionnaire de paquet.
Ainsi sous ubuntu on tapera simplement **sudo apt-get install wireshark**

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

2. Pourquoi utiliser a wireshark

Wireshark offre de nombreux avantages pour les débutants en cybersécurité et les professionnels du réseau. Voici quelques raisons pour lesquelles vous devriez considérer l'utilisation de Wireshark :

- **Identification des problèmes de réseau** : Wireshark vous permet d'examiner le trafic réseau pour détecter les problèmes de performances, les pannes, les congestions et les erreurs de configuration.
  Il vous aide à localiser les goulots d'étranglement et à résoudre les problèmes rapidement.

- **Analyse des menaces** : Wireshark peut être utilisé pour analyser le trafic malveillant et détecter les attaques réseau telles que les attaques par déni de service (DDoS), les attaques de phishing, les tentatives d'intrusion. Cela vous permet de renforcer la sécurité de votre réseau et de prendre des mesures préventives.

- **Apprentissage des protocoles réseau** : En utilisant Wireshark, vous pouvez étudier les protocoles réseau tels que TCP/IP, DNS, HTTP, etc. Cela vous donne une compréhension approfondie du fonctionnement des réseaux et vous aide à mieux diagnostiquer les problèmes.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

3. Comment Wireshark fonctionne-t-il?

- Wireshark utilise des **interfaces** de capture pour intercepter et enregistrer le trafic réseau.

Il peut capturer des paquets de données à partir de différents types de médias, tels que les connexions Ethernet filaires, les réseaux sans fil et les interfaces Bluetooth.

- Une fois les paquets capturés, Wireshark fournit des fonctionnalités d'analyse avancées pour filtrer, trier et afficher les données en fonction de différents critères.

Avec les filtres Wireshark, vous pouvez extraire des informations spécifiques à partir du trafic réseau capturé.

Par exemple, vous pouvez filtrer les paquets en fonction de l'adresse IP source ou de destination, du port, du protocole, etc.

Cela vous permet de cibler les paquets pertinents et d'obtenir des informations précises.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

4. Dans quels cas professionnels utilise-t-on Wireshark ?

- Prenons l'exemple d'une entreprise qui soupçonne une fuite d'informations confidentielles.

Dans ce cas, un professionnel de la sécurité pourrait utiliser Wireshark pour capturer le trafic réseau et analyser les paquets de données à la recherche d'activités suspectes.

En utilisant Wireshark, l'analyste peut filtrer les paquets en fonction de critères spécifiques tels que les adresses IP source et destination, les ports utilisés, les protocoles réseau, etc.

Cela permet de se concentrer uniquement sur le trafic pertinent et d'identifier les éventuelles communications suspectes ou non autorisées.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

4. Dans quels cas professionnels utilise-t-on Wireshark ?

L'analyste peut également utiliser les fonctionnalités avancées de Wireshark pour reconstruire les conversations réseau, suivre les sessions TCP, analyser les en-têtes de protocole et inspecter le contenu des paquets.

Cela lui permet de comprendre comment les données sont échangées, de vérifier si des informations sensibles sont transmises de manière sécurisée et d'identifier les éventuelles fuites d'informations.

En résumé, Wireshark est un outil essentiel dans un cas concret professionnel pour la détection des anomalies, la résolution des problèmes de réseau et l'investigation des potentielles activités malveillantes.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

5. Pour quel métier utilise-t-on Wireshark ?

Les Ethical Hackers, également connus sous le nom de **hackers éthiques** ou pentesters, utilisent Wireshark dans le cadre de leurs activités pour évaluer la sécurité des systèmes, réseaux et applications.

Voici comment Wireshark est utilisé dans le domaine de l'éthique du hacking :

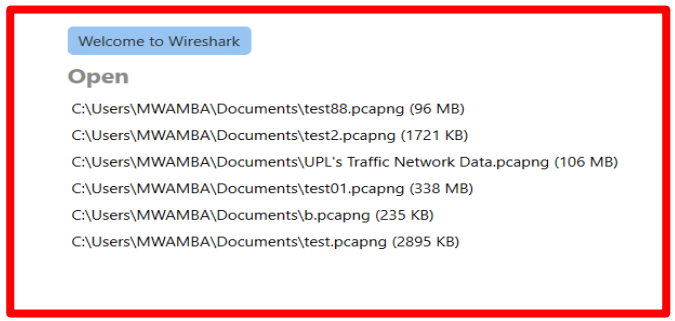
- ✓ **Évaluation de la vulnérabilité du réseau :** Les Ethical Hackers utilisent Wireshark pour analyser le trafic réseau, détecter les faiblesses et les vulnérabilités potentielles.
  En capturant et en analysant les paquets de données, ils peuvent identifier les ports ouverts, les services actifs et les éventuelles vulnérabilités qui pourraient être exploitées par des attaquants malveillants.

- ✓ **Détection des attaques et des intrusions :** Wireshark est un outil précieux pour détecter les attaques réseau telles que les tentatives d'intrusion, les attaques par déni de service (DDoS), les attaques de phishing, etc.
  Les Ethical Hackers utilisent les fonctionnalités avancées de Wireshark pour examiner les paquets et les flux de données, afin d'identifier les comportements anormaux ou les schémas suspects.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

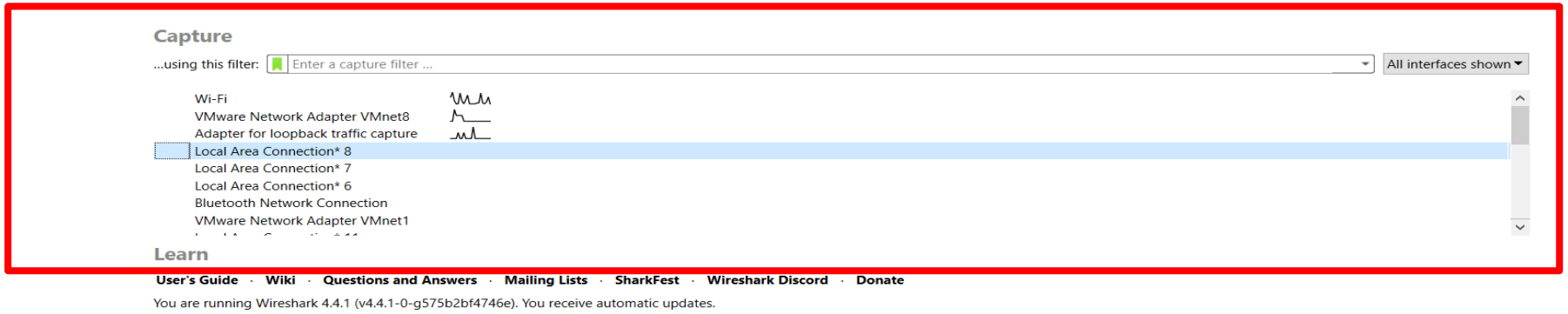
6. Lancement de Wireshark

Wireshark permet d'analyser un trafic enregistré dans un fichier annexe, mais également et surtout le trafic en direct sur des interfaces réseau. Cette seconde fonction nécessite de posséder les droits administrateurs, ou d'appartenir à un groupe poss édant ces droits



Analyse d'une capture précédente enregistrée sur fichier

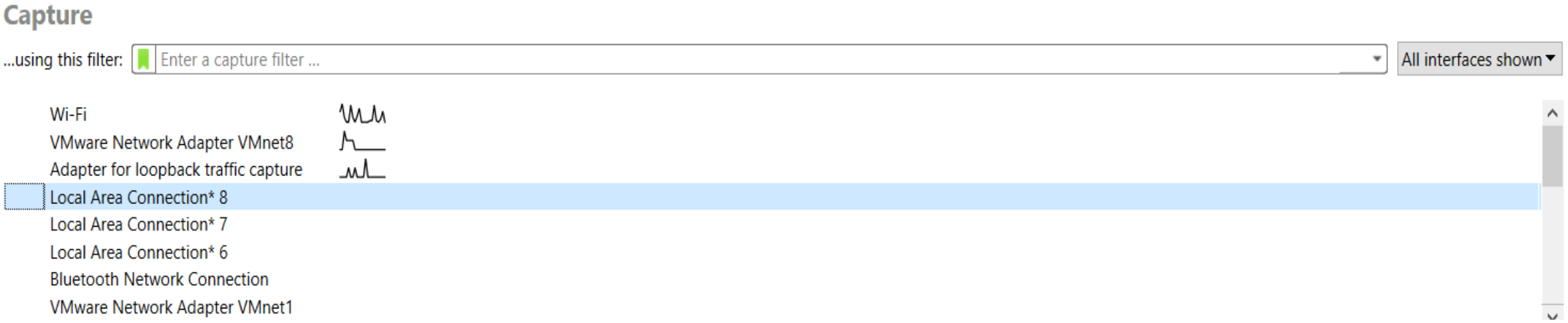
liste des interfaces et lancement rapide d'une capture



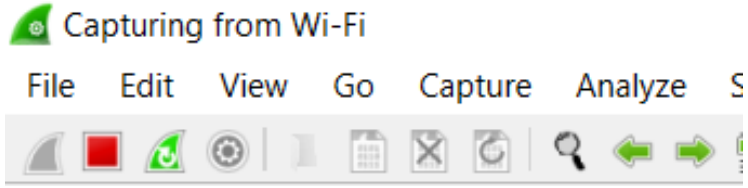
CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

7. Capture des trames sur le réseau

La principale utilisation que nous ferons de Wireshark consistera en la capture de trames réseau en live.
Pour lancer une capture live, plusieurs méthodes s'offrent à nous, parmi lesquelles :
➔ Cliquer directement sur l'interface désirée listée sur de Figure ci-dessous,. On cliquera par exemple sur le bouton « Wi-Fi» pour lancer la capture.

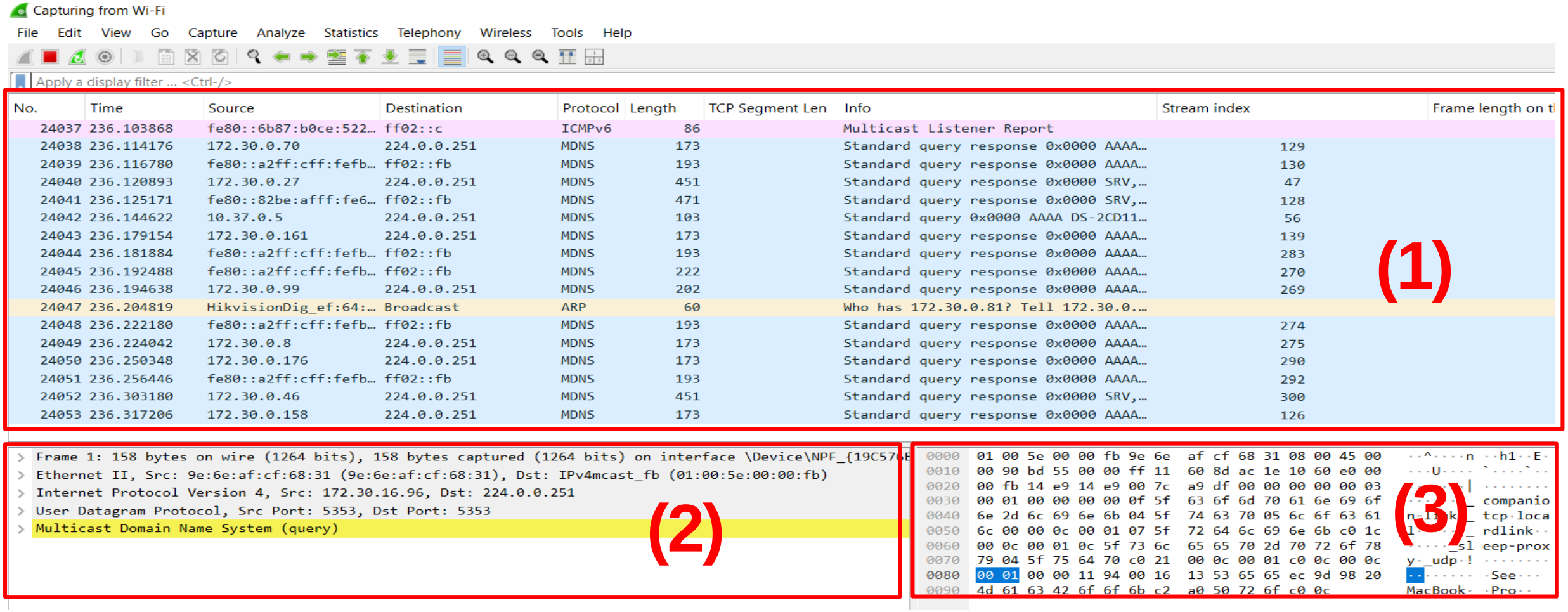


Une fois lancée, la capture peut être interrompue en cliquant que le 2ème bouton de la barre d'icônes (en partant de la gauche).



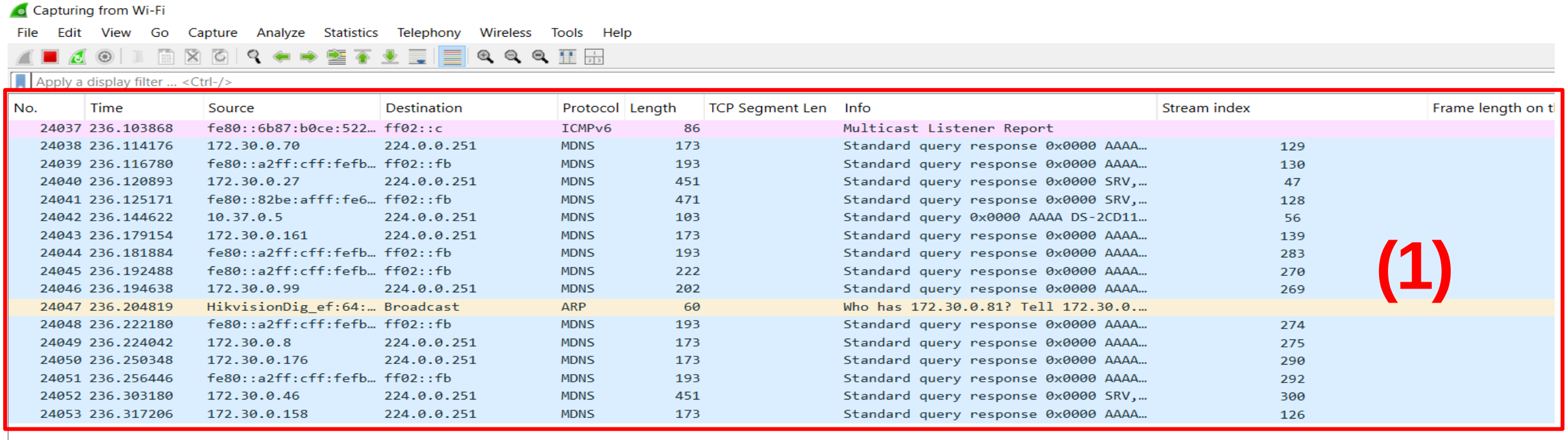
CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

7. Capture des trames sur le réseau



CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

7. Capture des trames sur le réseau



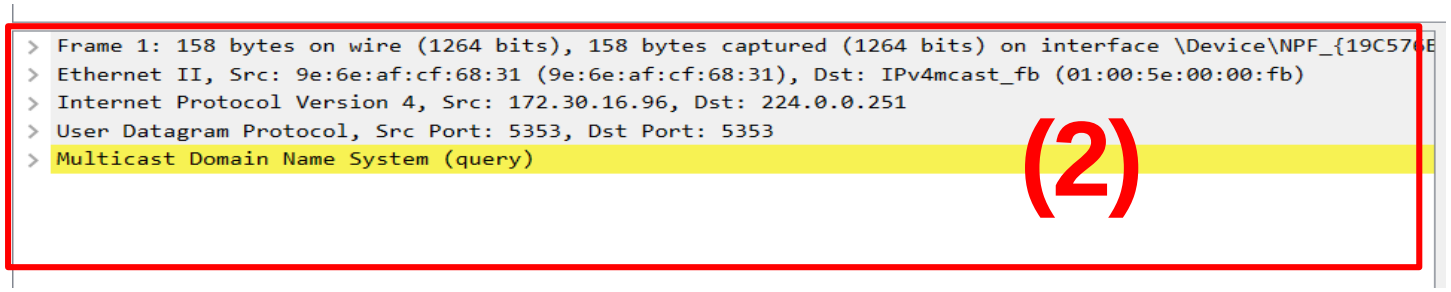
L'interface de l'analyseur est découpé en trois zone :

1. **Zone supérieure**, numérotée (1) sur Figure : liste l'ensemble des paquets capturés

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

7. Capture des trames sur le réseau

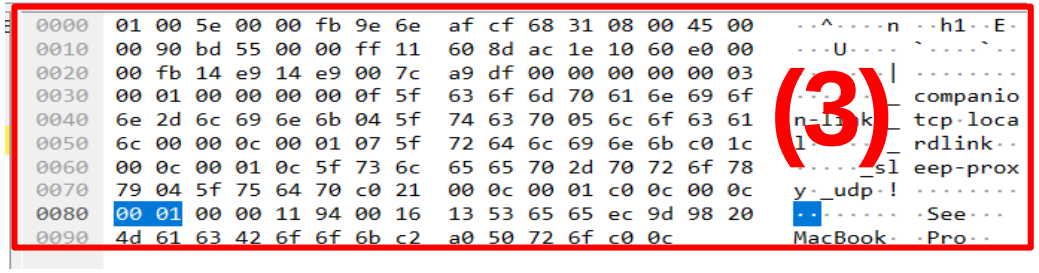
2. **Zone centrale**, numérotée (2) sur Figure : affiche le détail d'un paquet sélectionné dans la liste des paquets de la zone supérieure.
Les informations présentées y sont de loin les plus pertinentes, puisqu'il est possible de visualiser aisément les différents en-têtes résultant de l'encapsulation d'un message.



7. Capture des trames sur le réseau

3. **Zone inférieure**, numérotée (3) sur Figure : présente l'ensemble du paquet sous forme octale et ASCII.

Ces octets contiennent les en-têtes des différentes couches de l'architecture TCP/IP ainsi que les données transmises par le processus à l'origine du message.



CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

Encapsulation d'un paquet

Lorsqu'un paquet est sélectionné, la **zone centrale** permet de visualiser clairement les différentes couches d'encapsulation du paquet.

Par exemple si l'on sélectionne un paquet de type UDP , la zone centrale pourrait afficher quelque chose de similaire à ce qui est présenté Figure.

Les 5 entrées présentées correspondent à différentes encapsulations, ordonnées de la couche la plus basse à la couche la plus haute :

```
> Frame 16: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{19C576B2-8
> Ethernet II, Src: Routerboardc_5a:9b:88 (78:9a:18:5a:9b:88), Dst: LiteonTechno_d1:f4:93 (80:30:49:d1:f4:
> Internet Protocol Version 4, Src: 216.239.34.180, Dst: 172.30.16.146
> User Datagram Protocol, Src Port: 443, Dst Port: 62786
> Data (24 bytes)
```

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Encapsulation d'un paquet

Les 5 entrées présentées correspondent à différentes encapsulations, ordonnées de la couche la plus basse à la couche la plus haute :

```
> Frame 16: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{19C576B2-8
> Ethernet II, Src: Routerboardc_5a:9b:88 (78:9a:18:5a:9b:88), Dst: LiteonTechno_d1:f4:93 (80:30:49:d1:f4:
> Internet Protocol Version 4, Src: 216.239.34.180, Dst: 172.30.16.146
> User Datagram Protocol, Src Port: 443, Dst Port: 62786
> Data (24 bytes)
```

1. Données sur le média de capture : Wire = filaire sur Figure
2. Trame relative à la couche liaison de donnée : Ethernet II sur Figure
3. Paquet relatif à la couche réseau : Internet Protocol sur Figure
4. Datagramme relatif à la couche transport : User Datagram Protocol sur Figure
5. Données de l'application : regroupe généralement les couches session, présentation, application.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Détail de chaque niveau d'encapsulation

```
> Frame 16: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{19C576B2-8
> Ethernet II, Src: Routerboardc_5a:9b:88 (78:9a:18:5a:9b:88), Dst: LiteonTechno_d1:f4:93 (80:30:49:d1:f4:
v Internet Protocol Version 4, Src: 216.239.34.180, Dst: 172.30.16.146
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x80 (DSCP: CS4, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x0000 (0)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 56
    Protocol: UDP (17)
    Header Checksum: 0x89e5 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 216.239.34.180
    Destination Address: 172.30.16.146
    [Stream index: 1]
v User Datagram Protocol, Src Port: 443, Dst Port: 62786
    Source Port: 443
    Destination Port: 62786
    Length: 32
    Checksum: 0x53ac [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
    [Stream Packet Number: 3]
  > [Timestamps]
    UDP payload (24 bytes)
v Data (24 bytes)
    Data: 417b71fd05d73afb1568a36841d51f9743c48d2d6a9fb296
    [Length: 24]
```

Pour tout item correspondant à un niveau d'encapsulation, un clic sur le triangle en début deligne permet de dérouler l'en-tête afin de voir l'ensemble des champs le composant.

Certains champs peuvent également être déroulés.

Sur l'exemple présenté en Figure, nous avons étendu les entrées correspondant aux couches réseau, transport et application en cliquant sur les triangles correspondants.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Détail de chaque niveau d'encapsulation

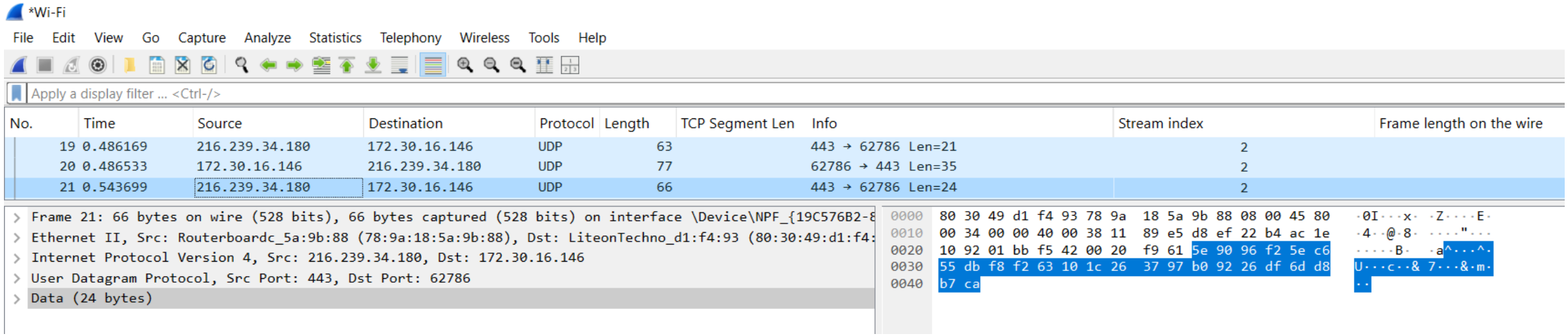
```
> Frame 16: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\
> Ethernet II, Src: Routerboardc_5a:9b:88 (78:9a:18:5a:9b:88), Dst: LiteonTechno_d1:f4:93 (
v Internet Protocol Version 4, Src: 216.239.34.180, Dst: 172.30.16.146
    0100 .... = Version: 4 (2)
    .... 0101 = Header Length: 20 bytes (5) (3)
  > Differentiated Services Field: 0x80 (DSCP: CS4, ECN: Not-ECT)
    Total Length: 52 (4)
    Identification: 0x0000 (0)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 56
    Protocol: UDP (17) (5)
    Header Checksum: 0x89e5 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 216.239.34.180 (6)
    Destination Address: 172.30.16.146 (7)
    [Stream index: 1]
v User Datagram Protocol, Src Port: 443, Dst Port: 62786
    Source Port: 443
    Destination Port: 62786
    Length: 32 (8)
    Checksum: 0x53ac [unverified]
    [Checksum Status: Unverified]
    [Stream index: 2]
    [Stream Packet Number: 3]
  > [Timestamps]
    UDP payload (24 bytes)
v Data (24 bytes) (9)
    Data: 417b71fd05d73afb1568a36841d51f9743c48d2d6a9fb296
    [Length: 24]
```

- La taille du Frame est de 66 octets soit 528 bits : **ref (1).**
- Le paquet est de type IP v4 : **ref (2)** sur Figure.
- La taille des en-têtes du paquet IP est de 20 octets : **ref (3).**
- Le paquet IP contient un en-tete (20 octets) ainsi que le datagramme UDP (32 octets). Sa taille totale est de 52 octets, taille rappelée en **ref (4)**
- Le type de données de ce paquet IP est un datagramme UDP : **ref (5)**
- L'IP de la machine source est 216.239.34.180 : **ref (6).**
- L'ip de la machine destination est 193.52.236.247 : **ref (7).**
- La taille totale du datagramme UDP est de 32 octets - **ref (8).**
- La taille des données envoyée par le processus est de 23 octets : **ref (9).**

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Visualisation octale de la trame



- La zone inférieure permet de visualiser la trame capturée sous forme octale. Un clic sur n'importe lequel des niveaux d'encapsulation permet de visualiser la portion d'octets correspondante dans la zone inférieure de l'analyseur.

- Pour n'importe quel niveau, un clic sur une valeur de champs permet de visualiser la portion d'octets correspondant à cette valeur dans le paquet au niveau de la zone inférieure de l'analyseur.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Filtrage de paquets

➢ **Principe et mise en place d'un filtre**

L'intégralité des paquets capturés est listée dans la zone supérieure de l'analyseur.
Il est souvent utile de **filtrer** les paquets à capturer, afin de pouvoir visualiser correctement un certain type de paquets seulement.

Wireshark permet de filtrer les paquets à capturer en fonction des informations des différentes couches d'encapsulation.

La mise en place d'un filtre s'effectue par le biais d'une règle de filtre à définir dans la zone « **filtre** » de l'analyseur.

Une règle de filtre est constituée d'un ensemble de **tests d'expressions** impliquant des noms de champs et des valeurs.

Un paquet n'est alors listé qu'à la condition qu'il satisfasse les conditions du filtre.

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Filtrage de paquets

➢ **Principe et mise en place d'un filtre**

L'ensemble des champs utilisables dans l'établissement des règles est listé dans la fenêtre pop-up accessible en cliquant sur le bouton « expression ».
Les règles peuvent être élaborées en sélectionnant les champs a partir de cette fenêtre , ou en les écrivant dire ctement dans la zone de filtre.

Un ensemble de filtres pré-définis est accessible en cliquant sur le bouton « **filter** ».

Cette liste de filtres peut être complétée d'entrées enregistrées.

Une fois le filtre défini, il ne faut pas oublier de l'appliquer avec le bouton « ***Apply*** »

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

8. Analyse d'un paquet

❖ Filtrage de paquets

➢ **Quelques filtres**

Désignation	**Filtre associé :**
Segments TCP uniquement	**tcp**
Paquets relatifs à TCP uniquement	**ip.proto == 0x06**
Adresse ip 192.168.0.1 ou 192.168.1.5	**ip.addr == 192.168.0.1 || ip.addr == 192.168.1.5**
Trafic HTTP uniquement	**http**
Segment TCP sauf sur port 80	**tcp && !(tcp.port == 80)**
Adresse Ethernet 00:FF:12:34:AE:FF	**eth.addr == 00:FF:12:34:AE:FF**
Trafic 192.168.0.1 vers 197.168.10.5	**ip.src == 192.168.0.1 && ip.dst == 197.168.10.5**
Trafic UDP entre ports 40 et 67	**udp && udp.port >= 40 && udp.port <=67**
Trafic MSN	**tcp && tcp.port ==1863**

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

	A	B	C	D	E	F	G	H	I	J
1	No.	Time	Source	Destination	Protocol	Length	TCP Segment Len	Info	Stream index	Frame length on the wire
2	1	0	172.30.16.99	51.116.246.106	TCP	54	0	61552 > 443 [ACK]	0	54
3	2	0.021027	172.30.16.13	172.30.63.255	MAC-Telnet	64		84:c5:a6:56:b7:e7	0	64
4	3	0.025159	172.30.16.13	172.30.63.255	MAC-Telnet	64		84:c5:a6:56:b7:e7	0	64
5	4	0.0334	172.30.16.99	8.8.8.8	DNS	77		Standard query 0x5	1	77
6	5	0.033568	172.30.16.99	8.8.8.8	DNS	77		Standard query 0x2	2	77
7	6	0.034381	172.30.16.13	172.30.63.255	MAC-Telnet	64		84:c5:a6:56:b7:e7	0	64
8	7	0.03629	172.30.16.13	172.30.63.255	MAC-Telnet	64		84:c5:a6:56:b7:e7	0	64
9	8	0.037312	172.30.16.13	172.30.63.255	MAC-Telnet	64		84:c5:a6:56:b7:e7	0	64
10	9	0.038304	172.30.16.13	172.30.63.255	MAC-Telnet	64		84:c5:a6:56:b7:e7	0	64
11	10	0.040466	172.30.0.46	224.0.0.251	MDNS	202		Standard query res	3	202
12	11	0.043007	fe80::a2ff:cff:fefb:2cc8	ff02::fb	MDNS	222		Standard query res	4	222
13	12	0.044908	172.30.0.124	224.0.0.251	MDNS	171		Standard query res	5	171
14	13	0.04699	fe80::2632:aeff:feae:7aa3	ff02::fb	MDNS	191		Standard query res	6	191
15	14	0.058788	172.30.13.208	239.255.255.250	UDP	77		40180 > 15600 Le	7	77
16	15	0.059852	fe80::57:24d3:173:dfd2	ff02::2:ff73:386e	ICMPv6	86		Multicast Listener Report		86
17	16	0.061102	fe80::57:24d3:173:dfd2	ff02::1:ff73:dfd2	ICMPv6	86		Multicast Listener Report		86
18	17	0.062609	fd24:da33:7381:cc00:8ad	ff02::fb	MDNS	127		Standard query 0x0	8	127
19	18	0.063418	8.8.8.8	172.30.16.99	DNS	295		Standard query res	1	295
20	19	0.063418	8.8.8.8	172.30.16.99	DNS	251		Standard query res	2	251
21	20	0.064615	172.30.16.99	18.197.70.228	TCP	66	0	61553 > 443 [SYN]	1	66

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 1 : Importer les packages nécessaires et de la dataset

```
[429]: 1 import numpy as np
       2 import pandas as pd
       3 import matplotlib.pyplot as plt
```

```
[430]: 1 traffic_data = pd.read_csv('UPL_traffic_reseau.csv', encoding = 'unicode_escape')
```

```
[431]: 1 traffic_data
```

Code Python où on utilises les bibliothèques importées pour appliquer la méthode Elbow avec l'algorithme k-means clustering.

	No.	Time	Source	Destination	Protocol	Length	TCP Segment Len	Info	Stream index	Frame length on the wire
0	1	0.000000	172.30.16.99	51.116.246.106	TCP	54	0.0	61552 > 443 [ACK] Seq=1 Ack=1 Win=515 Len=0	0.0	54
1	2	0.021027	172.30.16.13	172.30.63.255	MAC-Telnet	64	NaN	84:c5:a6:56:b7:e7 > 78:9a:18:5a:9b:7f Directio...	0.0	64
2	3	0.025159	172.30.16.13	172.30.63.255	MAC-Telnet	64	NaN	84:c5:a6:56:b7:e7 > 78:9a:18:5a:9b:7f Directio...	0.0	64
3	4	0.033400	172.30.16.99	8.8.8.8	DNS	77	NaN	Standard query 0x5fcc A acns.my.avira.com	1.0	77
4	5	0.033568	172.30.16.99	8.8.8.8	DNS	77	NaN	Standard query 0x2681 AAAA acns.my.avira.com	2.0	77

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 2 : Étape de prétraitement des données

```
1 traffic_data.shape
```

```
(155781, 10)
```

```
1 traffic_data.columns
```

```
Index(['No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length',
       'TCP Segment Len', 'Info', 'Stream index', 'Frame length on the wire'],
      dtype='object')
```

```
1 traffic_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155781 entries, 0 to 155780
Data columns (total 10 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   No.                        155781 non-null  int64
 1   Time                       155781 non-null  float64
 2   Source                     155781 non-null  object
 3   Destination                155781 non-null  object
 4   Protocol                   155781 non-null  object
 5   Length                     155781 non-null  int64
 6   TCP Segment Len            57891 non-null   float64
 7   Info                       155781 non-null  object
 8   Stream index               144134 non-null  float64
 9   Frame length on the wire   155781 non-null  int64
dtypes: float64(3), int64(3), object(4)
memory usage: 11.9+ MB
```

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 2 : Étape de prétraitement des données

```
1 traffic_data['Source'].unique()
```

```
array(['172.30.16.99', '172.30.16.13', '172.30.0.46',
       'fe80::a2ff:cff:fefb:2cc8', '172.30.0.124',
       'fe80::2632:aeff:feae:7aa3', '172.30.13.208',
       'fe80::57:24d3:173:dfd2',
       'fd24:da33:7381:cc00:8adc:97ff:fe23:4b5b', '8.8.8.8',
       '8e:dc:97:23:5a:a1', '172.30.0.6', 'fe80::a2ff:cff:fefb:2c5f',
       '172.30.0.33', 'fe80::a2ff:cff:fefb:2cfa', 'HikvisionDig_1b:ad:8a',
       'HikvisionDig_1a:2c:96', '162.254.196.79', '162.254.196.80',
       '172.30.0.3', 'fe80::a2ff:cff:fefb:2c64', '88:dc:97:23:43:0f',
       '172.30.1.72', '0.0.0.0', 'Routerboardc_5a:9b:88', '18.197.70.228',
       'Intel_56:b7:e7', '172.30.0.19', 'fe80::a2ff:cff:fefb:2cdd',
       'fe80::8adc:97ff:fe23:5032', '88:dc:97:23:50:53', '172.30.0.187',
       'fe80::2632:aeff:feaa:c91d', '88:dc:97:23:4e:0a', '172.30.0.97',
       'fe80::a2ff:cff:fefb:2c68', '172.30.0.133',
       'fe80::2632:aeff:feaa:c952', '172.30.0.159',
       'fe80::a2ff:cff:fefb:2cb6', '172.30.0.72',
       'HikvisionDig_aa:c9:58', '172.30.1.69',
       'fe80::5c9f:43e9:631e:db30', '34.0.245.166',
       'HikvisionDig_aa:c9:1d', 'HikvisionDig_aa:c9:15',
       'HikvisionDig_ae:7b:3b', '88:dc:97:23:4d:80', '88:dc:97:23:4c:2a',
       '88:dc:97:23:42:e2', '88:dc:97:23:4d:77', 'HikvisionDig_aa:c9:61',
       'HikvisionDig_aa:c9:67', 'HikvisionDig_ae:7a:6d', '95.100.108.201',
       '95.100.108.160', 'HikvisionDig_aa:c9:2e', '172.30.0.35',
       '172.30.0.75', 'HikvisionDig_aa:c9:4b',
       'fe80::8adc:97ff:fe23:5b69', '3.121.120.125',
       'HikvisionDig_ae:7a:4b', '20.66.110.161', 'HikvisionDig_aa:c9:6b',
       'HikvisionDig_aa:c9:47', '158.23.16.71', '34.204.28.63',
       '172.30.0.25', '172.30.0.15', '172.30.0.5',
       'HikvisionDig_aa:c9:3a', '172.202.65.254', '20.42.72.131',
       'HikvisionDig_aa:c9:34', '20.106.94.33', 'HikvisionDig_aa:c9:40',
       '204.79.197.222', 'HikvisionDig_aa:c9:39', 'HikvisionDig_ae:7a:82',
       '3.215.11.190', '35.213.145.237', '172.30.0.85', '172.30.0.45',
       '88:dc:97:23:43:6f', 'HikvisionDig_aa:c9:10',
```

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 2 : Étape de prétraitement des données

```
1 traffic_data['Source'].value_counts()
```

```
Source
2.22.201.145              24549
172.30.16.99              23508
172.30.16.13              14381
192.178.54.14              4297
2.17.165.4                 3719
                          ...
88:dc:97:23:44:32             1
ee:6d:17:7c:94:50             1
88:dc:97:23:3f:0a             1
88:dc:97:23:5b:69             1
SamsungElect_e7:c4:2a         1
Name: count, Length: 897, dtype: int64
```

```
1 traffic_data['Destination'].value_counts()
```

```
Destination
172.30.16.99              44705
224.0.0.251               29632
ff02::fb                  29126
172.30.63.255             14713
2.22.201.145              12080
                          ...
ff02::1:fffb:92ad             2
173.194.76.188                1
162.254.196.79                1
51.116.246.106                1
ff02::1:ff93:6dda             1
Name: count, Length: 203, dtype: int64
```

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 2 : Étape de prétraitement des données

```
1 traffic_data['Protocol'].unique()
```

```
array(['TCP', 'MAC-Telnet', 'DNS', 'MDNS', 'UDP', 'ICMPv6', 'EAPOL',
       'ARP', 'SSDP', 'DHCP', 'TLSv1.2', '0x8033', 'IPv4', 'DHCPv6',
       'QUIC', 'TLSv1.3', 'STEAMDISCOVER', '0x9998', 'HTTP', 'MNDP',
       'IGMPv3', 'NBNS', 'LLMNR', 'PKIX-CRL', 'SSL', 'HTTP/XML', 'RK512',
       'BROWSER', 'UDP/XML', 'OCSP', 'HCrt', 'HTTP/JSON', 'IGMPv2'],
      dtype=object)
```

```
1 traffic_data.drop(['No.','Time', 'Source','Destination','Info'], axis='columns', inplace=True)
```

```
1 traffic_data
```

	Protocol	Length	TCP Segment Len	Stream index	Frame length on the wire
0	TCP	54	0.0	0.0	54
1	MAC-Telnet	64	NaN	0.0	64
2	MAC-Telnet	64	NaN	0.0	64
3	DNS	77	NaN	1.0	77
4	DNS	77	NaN	2.0	77

```
1 traffic_data.drop('Protocol', axis=1, inplace=True)
```

```
1 traffic_data.head()
```

	Length	TCP Segment Len	Stream index	Frame length on the wire
0	54	0.0	0.0	54
1	64	NaN	0.0	64
2	64	NaN	0.0	64
3	77	NaN	1.0	77
4	77	NaN	2.0	77

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 2 : Étape de prétraitement des données

```
1 traffic_data.isnull().sum()
```

```
Length                          0
TCP Segment Len             97890
Stream index                11647
Frame length on the wire        0
dtype: int64
```

```
1 #traffic_data.dropna(inplace=True)
```

```
1 traffic_data['TCP Segment Len'].fillna(traffic_data['TCP Segment Len'].mean(), inplace=True)
2 traffic_data['Stream index'].fillna(traffic_data['Stream index'].mean(), inplace=True)
```

	Length	TCP Segment Len	Stream index	Frame length on the wire
0	54	0.000000	0.0	54
1	64	879.724845	0.0	64
2	64	879.724845	0.0	64
3	77	879.724845	1.0	77
4	77	879.724845	2.0	77

```
1 traffic_data.isnull().sum()
```

```
Length                          0
TCP Segment Len                 0
Stream index                    0
Frame length on the wire        0
dtype: int64
```

```
1 traffic_data.shape
```

```
(155781, 4)
```

```
1 x = traffic_data.iloc[0:1500,2:].values
2 x
```

```
array([[   0.,   54.],
       [   0.,   64.],
       [   0.,   64.],
       ...,
       [   8.,   54.],
       [   8., 1510.],
       [   8., 1510.]])
```

CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

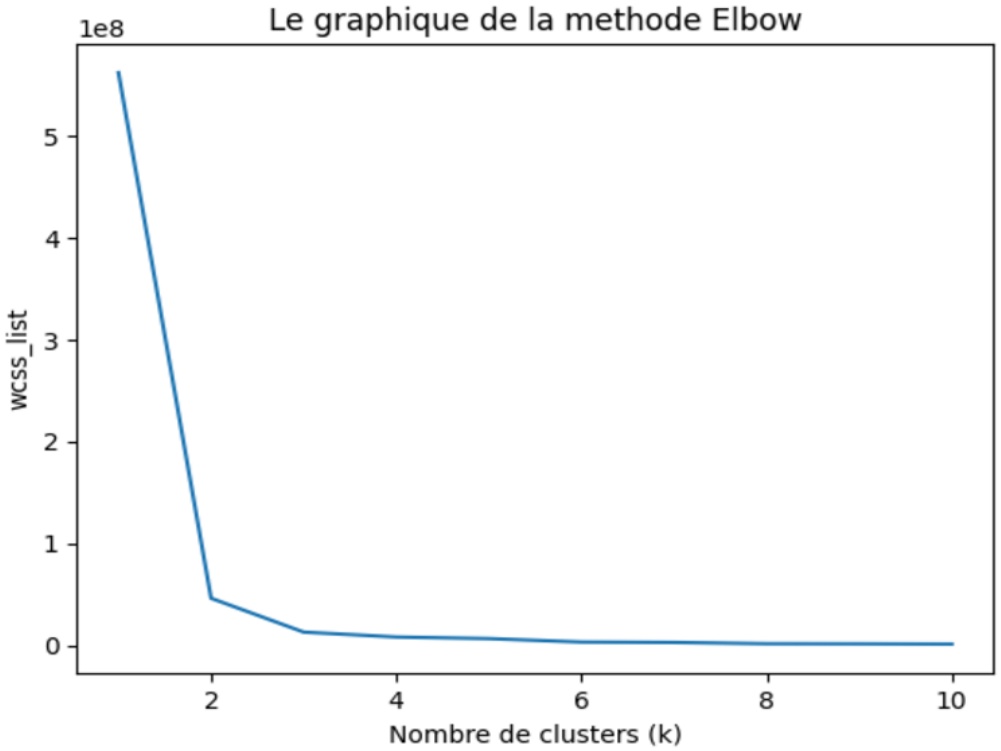
Implémentation Python de l'algorithme de clustering K-means

Étape 3 : Trouver le nombre optimal de clusters à l'aide de la méthode du coude

```
# Trouver le nombre optimal de clusters en utilisant la methode du coude
from sklearn.cluster import KMeans
```

```
wcss_list = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('Le graphique de la methode Elbow')
plt.xlabel('Nombre de clusters (k)')
plt.ylabel('wcss_list')
plt.show()
```



CHAPITRE 5 Analyse du trafic réseau pour le Machine Learning

9. Analyse de Traffic Reseau de l'UPL

Implémentation Python de l'algorithme de clustering K-means

Étape 4 : Entraînement de l'algorithme K-means sur les données d'entraînement

```
1 kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
```

```
1 kmeans.fit(x)
```

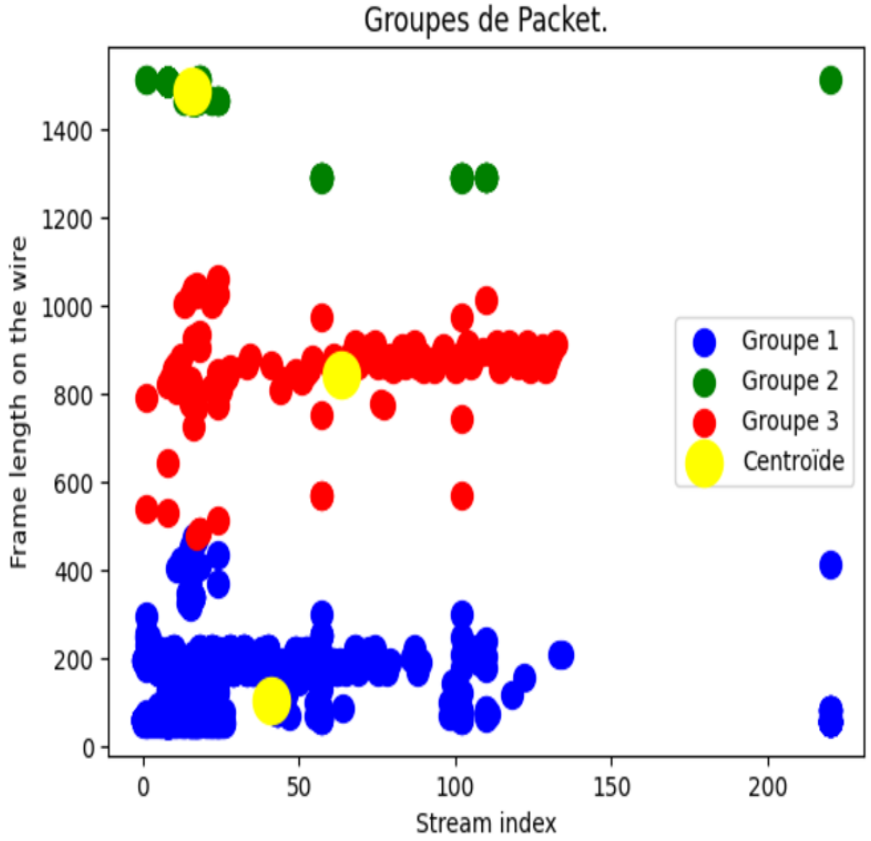
```
KMeans
KMeans(n_clusters=3, random_state=42)
```

```
1 y_predict = kmeans.predict(x)
```

```
1 y_predict
```

```
array([0, 0, 0, ..., 0, 1, 1], dtype=int32)
```

```
1  #Visualisation des clusters
2  plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Groupe 1') # Pour le premier groupe.
3  plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Groupe 2') # Pour le deuxième groupe.
4  plt.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Groupe 3') # Pour le troisième groupe.
5  plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroïde')
6  plt.title('Groupes de Packet.')
7  plt.xlabel('Stream index')
8  plt.ylabel('Frame length on the wire')
9  plt.legend()
10 plt.show()
```



Fin