INF3405 – Réseaux informatiques

Hiver 2020

TP No. 3

Groupe 2

[1928777] – Mariam Sarwat

[1935516] – Louis-Maxime Bois

Soumis à : Liliane-Caroline Demers

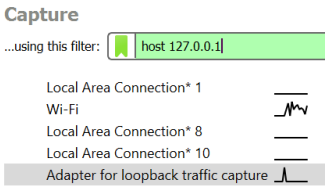
09/04/20

C) Analyse du flot de données de l'application "Traitement Sobel"

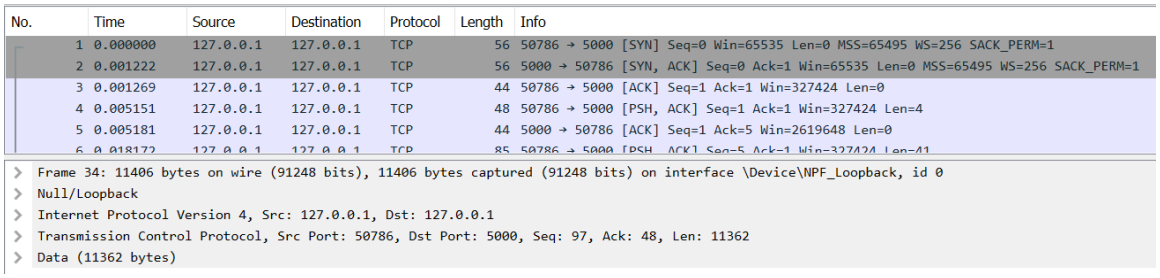
5.2 Appliquer un outil d'ingénierie

1) **Quel filtre appliqueriez-vous afin d'afficher uniquement les échanges entre le client et le serveur? (<u>1 point</u>)**

   Host 127.0.0.1

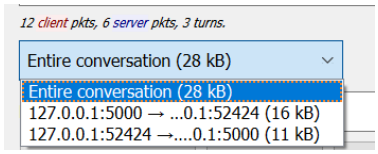


2) **À la lumière de vos observations, dites quel protocole de la couche 4 est utilisé pour la communication entre le client et le serveur. (<u>0.5 point</u>)**



   Le protocol de la couche 4 utilisé pour la communication entre le client et le serveur est le protocole **TCP**.

3) **Combien de paquets et d'octets de données ont été envoyés du client vers le serveur et du serveur vers le client ? (<u>2 points</u>)**



   On retrouve qu'il y a 18 paquets de donnée qui sont envoyés (28Koctets).

   <u>Client => Serveur</u>

   Il y a 12 paquets de donnée qui sont envoyés du client vers le serveur, soit 11 Koctets.

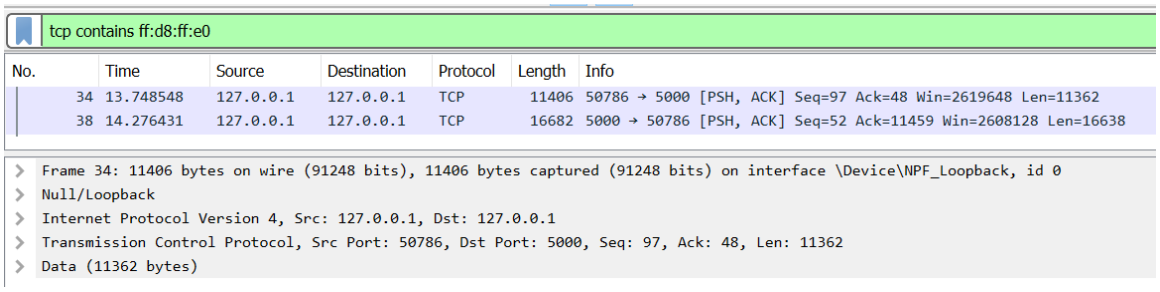
   <u>Serveur => Client</u>

   Il y a 8 paquets de donnée qui sont envoyés du serveur vers le client, soit 16 Koctets.

3.5 Analyser les résultats expérimentaux

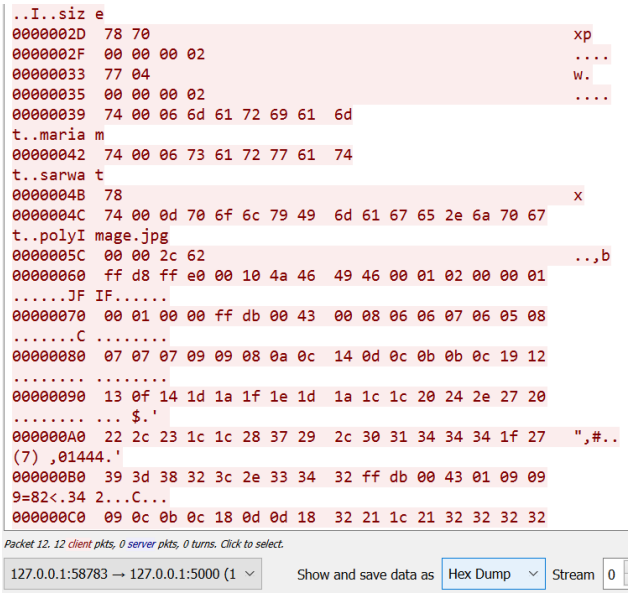
4) **Normalement, le standard IEEE 802.3 limite la taille d'une trame Ethernet à 1518 octets. Dans votre capture Wireshark, existe-t-il des paquets ayant une taille supérieure à 1518 octets? Si oui, expliquez pourquoi et comment ce paquet réussit à transiger sur le réseau alors que sa taille est plus grande que celle spécifiée par le standard. (2.5 points)**

Oui, il y a deux paquets de taille supérieur à 1518 octets (voir capture). En effet, c'est grâce au protocole Ethernet II que ces paquets on réussit à transiger. Comme leur taille est supérieure à la taille limite d'une trame Ethernet exigée par le standard IEEE 802.3, les deux paquets mentionnés ont été envoyés sous forme de trames Ethernet II.



5) **Quel type d'information êtes-vous capables d'extraire de Wireshark en lien avec l'authentification au serveur de traitement d'images? (<u>1 point</u>)**

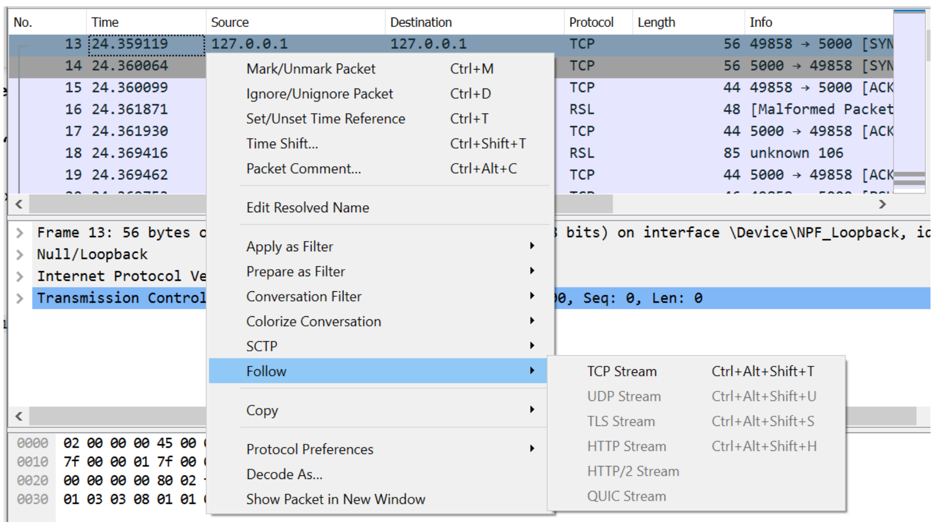
En effet, en affichant le contenu du « TCP stream » (soit le contenu des paquets), on retrouve le nom d'utilisateur et le mot de passe entré par le client afin qu'il s'authentique.



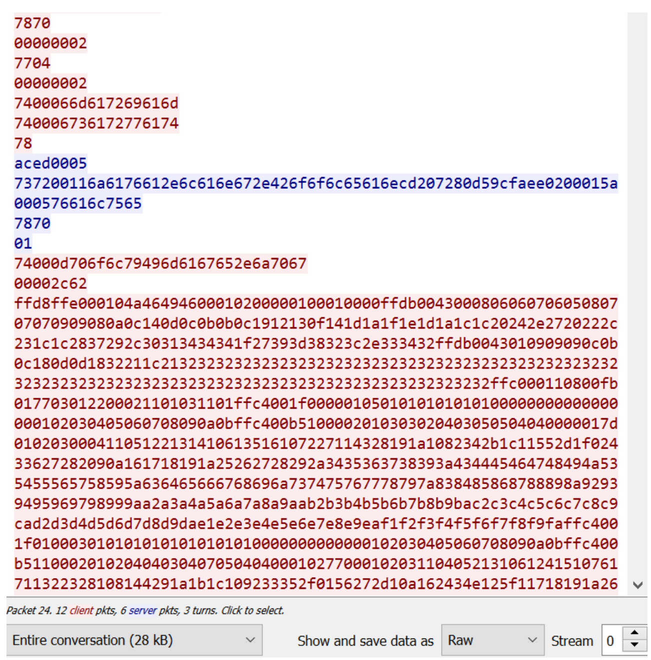
2.2 Explorer des approches de résolution et planifier la démarche

6) **Il est possible, avec Wireshark, d'extraire l'image envoyée par le client ou l'image traitée. Donnez les étapes à suivre, incluant des captures d'écran montrant chaque étape permettant l'extraction de l'image envoyée du client vers le serveur. Servez-vous des propriétés du fichier.jpg énoncées plus haut. Indice: utilisez le programme WinHex après avoir sauvegardé le flot de données en format " Raw " (<u>2 points</u>)**

   a. Faire un clic droit sur un paquet telle quelle –> choisir l'option « Follow » -> et puis l'option « TCP Stream ».

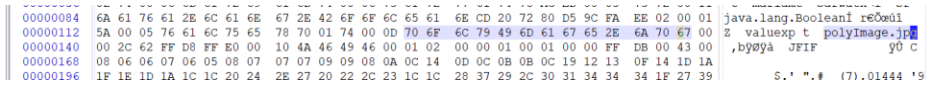


   b. Enregistrer le fichier sous le format « Raw »

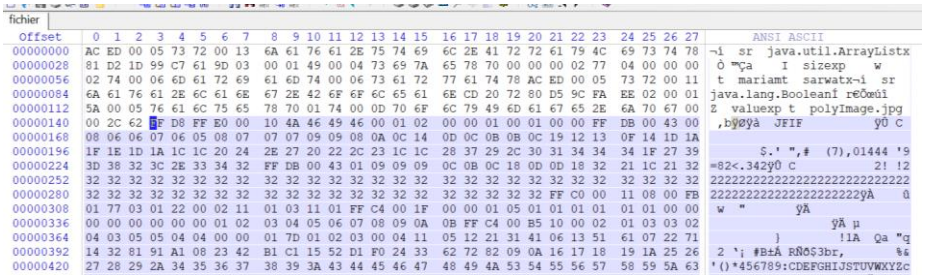


c. Ouvrir le fichier sauvegardé à l'étape ii dans WinHex

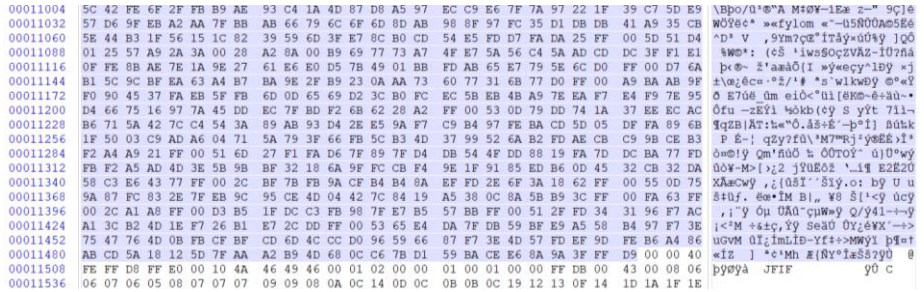
d. Retrouver le nom de l'image envoyée, soit polyImage.jpg (utiliser un Ctrl+F) –> le texte souligné est le nom du fichier à transformer.



e. Finalement, comme mentionné dans l'énoncé, les fichiers .jpg commence par FF D8 FF E0 et se termine avec FF D9. Alors, pour retrouver les données (de l'image) qui sont envoyé du client vers le serveur, il suffit rechercher les HEX « FF D8 FF E0 » pour le début et « FF D9 » pour la fin de l'extraction.



….



9.4 Évaluer les risques et les incertitudes d'une situation

7) **Suite à toute cette analyse que pouvez-vous conclure quant à la sécurité de l'application de traitement d'images que vous avez développé lors du travail pratique no.1 (<u>1 point</u>)**

À la suite de notre analyse, on conclut qu'il est très facile de ''sniffer'' les informations envoyées entre le client et le serveur. En effet, il est possible de récupérer toutes les données d'authentification des utilisateurs, par exemple. Bref, la sécurité de l'application développé lors du TP1 n'est pas sécuritaire.

9. Analyse d'une application client-serveur "secrète" (10 points)

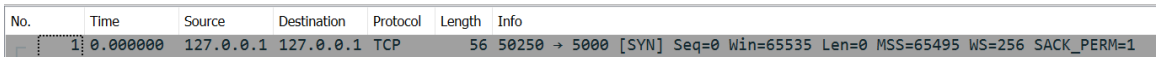
B) Mode secret (1, 2, 3 et 4) (2 points chaque)

3.6. Vérifier les hypothèses et argumenter

1) **Quel protocole de la couche transport est utilisé? Dans le cas de TCP, montrer le tout premier échange entre le client et le serveur lors de l'initialisation de la connexion, comment ce nomme cet échange? Dans le cas d'UDP, est-ce que ce même échange à lieu? Pourquoi? (<u>0.5 point</u>)**

<u>Mode 1 :</u>

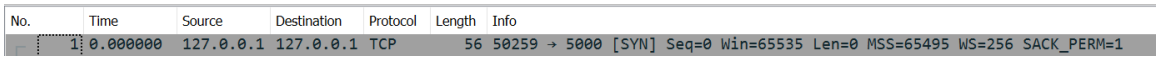
Le protocole utilisé est TCP. Le premier échange entre le client et le serveur se nomme « SYN » qui signifie « *Synchronize* ». En effet, cet échange est une demande pour établir une nouvelle connexion entre le client et le serveur.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	50250 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1

<u>Mode 2 :</u>

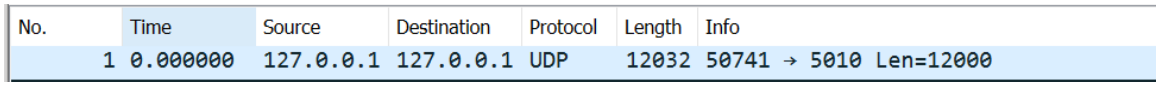
Le protocole utilisé est TCP. Le premier échange entre le client et le serveur se nomme « SYN » qui signifie « *Synchronize* ». En effet, cet échange est une demande pour établir une nouvelle connexion entre le client et le serveur.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	50259 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1

<u>Mode 3 :</u>

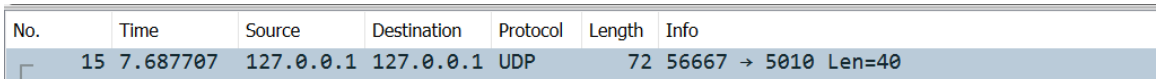
Le protocole utilisé est UDP. Alors, il n'y a pas de premier échange entre le client et le serveur, car UDP ne s'assure pas de la synchronisation. Effectivement, ce dernier envoie les paquets et c'est tout.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	12032	50741 → 5010 Len=12000

<u>Mode 4 :</u>

Le protocole utilisé est UDP. Alors, il n'y a pas de premier échange entre le client et le serveur, car UDP ne s'assure pas de la synchronisation. Effectivement, ce dernier envoie les paquets et c'est tout.

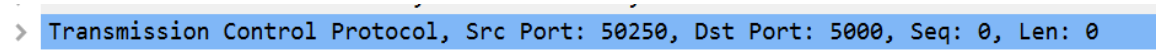


No.	Time	Source	Destination	Protocol	Length	Info
15	7.687707	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40

2) **En vous basant sur les informations recueillies par Wireshark, indiquez les ports source et destination utilisés par la couche 4. (<u>0.5 point</u>)**

<u>Mode 1 :</u>

Lors de la première transmission, on retrouve que le port source est 50250 et le port destination, quant à lui, est le 5000.



```
> Transmission Control Protocol, Src Port: 50250, Dst Port: 5000, Seq: 0, Len: 0
```

Mode 2 :

Lors de la première transmission, on retrouve que le port source est 50259 et le port destination, quant à lui, est le 5000.

```
> Transmission Control Protocol, Src Port: 50259, Dst Port: 5000, Seq: 0, Len: 0
```

Mode 3 :

On retrouve 50741 comme port source et 5010 comme port destination.

```
> User Datagram Protocol, Src Port: 50741, Dst Port: 5010
```

Mode 4 :

On retrouve 56667 comme port source et 5010 comme port destination.

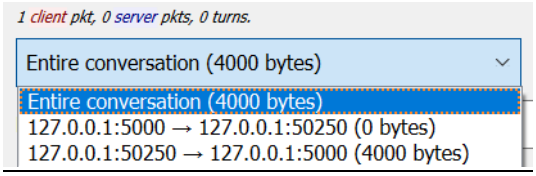
```
> User Datagram Protocol, Src Port: 56667, Dst Port: 5010
```

3) **Combien de paquets et d'octets contenant des données ont été envoyés par le client vers le serveur? Par le serveur vers le client? Montrer où vous avez trouvé cette information. (0.5 point)**

Mode 1 :

Client => Serveur : 1 paquet a été envoyé de 4000 octets.

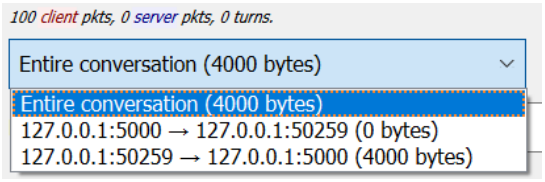
Serveur => Client : Aucun paquet a été envoyé, donc 0 octets.



Mode 2 :

Client => Serveur : 100 paquets ont été envoyés d'une taille totale de 4000 octets.

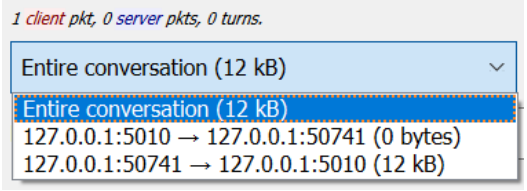
Serveur => Client : Aucun paquet a été envoyé, donc 0 octets.



Mode 3 :

Client => Serveur : 1 paquet a été envoyé de 12 Koctet.

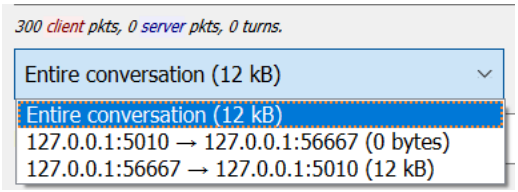
Serveur => Client : Aucun paquet a été envoyé, donc 0 octets.



Mode 4 :

Client => Serveur : 300 paquets ont été envoyés d'une taille totale de 12 Koctet.

Serveur => Client : Aucun paquet a été envoyé, donc 0 octets.



4) **À la lumière de votre analyse, que fait le client? Selon vous, combien d'itérations le client a-t-il faites pour envoyer ces données? (0.5 point)**

Mode 1 :

Le client commence, tout d'abords, avec l'envoie d'une demande de connexion au serveur avec [SYN]. Ce dernier, attend d'avoir reçu l'accusé de réception du serveur et envoie, à son tour, un accusé de réception au serveur ([ACK]). Par la suite, le client envoie les données en 1 seule itération et envoie un message de « acknowlegment ». Lorsque le client a terminé sa tâche, il envoie une demande au serveur pour mettre fin à la connexion avec [FIN, ACK]. Finalement, il envoie un accusé de réception au serveur avec [ACK] pour confirmer la fin de la communication.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	50250 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000936	127.0.0.1	127.0.0.1	TCP	56	5000 → 50250 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000982	127.0.0.1	127.0.0.1	TCP	44	50250 → 5000 [ACK] Seq=1 Ack=1 Win=327424 Len=0
4	0.002377	127.0.0.1	127.0.0.1	TCP	4044	50250 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=4000
5	0.002408	127.0.0.1	127.0.0.1	TCP	44	5000 → 50250 [ACK] Seq=1 Ack=4001 Win=2619648 Len=0
6	0.002452	127.0.0.1	127.0.0.1	TCP	44	50250 → 5000 [FIN, ACK] Seq=4001 Ack=1 Win=327424 Len=0
7	0.002464	127.0.0.1	127.0.0.1	TCP	44	5000 → 50250 [ACK] Seq=1 Ack=4002 Win=2619648 Len=0
8	0.003214	127.0.0.1	127.0.0.1	TCP	44	5000 → 50250 [FIN, ACK] Seq=1 Ack=4002 Win=2619648 Len=0
9	0.003261	127.0.0.1	127.0.0.1	TCP	44	50250 → 5000 [ACK] Seq=4002 Ack=2 Win=327424 Len=0

Mode 2 :

Le client commence, tout d'abords, avec l'envoie d'une demande de connexion au serveur avec [SYN]. Ce dernier, attend d'avoir reçu l'accusé de réception du serveur et envoie, à son tour, un accusé de réception au serveur ([ACK]). Ensuite, le client envoie 40 bytes de données (par paquets) et attend d'avoir reçu l'accusé de réception du serveur avant d'envoyer le prochain paquet. Ceci est répété pour 100 itérations. À la fin de l'envoie des données, le client envoie une demande au serveur pour mettre fin à la connexion avec [FIN, ACK]. Finalement, ce dernier envoie un accusé de réception au serveur avec [ACK] pour confirmer la fin de la communication.

1	0.000000	127.0.0.1	127.0.0.1	TCP	56	50259 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.001638	127.0.0.1	127.0.0.1	TCP	56	5000 → 50259 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.001736	127.0.0.1	127.0.0.1	TCP	44	50259 → 5000 [ACK] Seq=1 Ack=1 Win=327424 Len=0
4	0.002403	127.0.0.1	127.0.0.1	TCP	84	50259 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=40
5	0.002429	127.0.0.1	127.0.0.1	TCP	44	5000 → 50259 [ACK] Seq=1 Ack=41 Win=2619648 Len=0
6	0.002444	127.0.0.1	127.0.0.1	TCP	84	50259 → 5000 [PSH, ACK] Seq=41 Ack=1 Win=327424 Len=40
7	0.002453	127.0.0.1	127.0.0.1	TCP	44	5000 → 50259 [ACK] Seq=1 Ack=81 Win=2619648 Len=0

…

202	0.004453	127.0.0.1	127.0.0.1	TCP	84	50259 → 5000 [PSH, ACK] Seq=3961 Ack=1 Win=327424 Len=40
203	0.004460	127.0.0.1	127.0.0.1	TCP	44	5000 → 50259 [ACK] Seq=1 Ack=4001 Win=2615808 Len=0
204	0.004491	127.0.0.1	127.0.0.1	TCP	44	50259 → 5000 [FIN, ACK] Seq=4001 Ack=1 Win=327424 Len=0
205	0.004501	127.0.0.1	127.0.0.1	TCP	44	5000 → 50259 [ACK] Seq=1 Ack=4002 Win=2615808 Len=0
206	0.004527	127.0.0.1	127.0.0.1	TCP	44	5000 → 50259 [FIN, ACK] Seq=1 Ack=4002 Win=2615808 Len=0
207	0.004557	127.0.0.1	127.0.0.1	TCP	44	50259 → 5000 [ACK] Seq=4002 Ack=2 Win=2619648 Len=0

Mode 3 :

Le client envoie un seul paquet de donnée au serveur. Cela nécessite une seule itération.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	12032	50741 → 5010 Len=12000

Mode 4 :

Le client envoie 300 paquets de données au serveur. En suivant la même logique, ceci nécessite 300 itérations au total. Le client envoie simplement les paquets les uns après les autres.

…

498	7.695496	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
499	7.695510	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
500	7.695524	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
501	7.695538	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
502	7.695553	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
503	7.695567	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
504	7.695581	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
505	7.695596	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
506	7.695610	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
507	7.695624	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
508	7.695639	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
509	7.695653	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
510	7.695667	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
511	7.695681	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
512	7.695696	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40
513	7.695710	127.0.0.1	127.0.0.1	UDP	72	56667 → 5010 Len=40

C) Analyse des performances et protocole TCP (2 points)

1) **Comparez la performance des envois de données pour le mode 1 et le mode 2. Qu'est-ce qui diffère entre ces deux modes? Lequel est le plus performant selon vous et pourquoi? (0.5 point)**

En comparant ces deux modes, on aperçoit qu'ils envoient, touts les deux, du client vers le serveur des données d'une taille totale de 4000 bytes. Dans le mode 1, les données sont envoyées dans un seul et unique paquet de 4000 bytes. Dans ce cas, le client attend un accusé de réception unique à la fin de la transmission du paquet. Tandis que dans le mode 2, les données sont envoyées dans 100 différents paquets où chaque paquet possède une taille de 40 bytes. Dans ce cas, le client attend un accusé de réception du serveur avant d'envoyer un nouveau paquet, donc il attend 100 ACK. C'est pour cette raison qu'on croit que le mode 1 serait plus performante que le mode 2.

2) **Comparer la performance des envois de données pour le mode 3 et le mode 4. Qu'est-ce qui diffère entre ces deux modes? Lequel est le plus performant selon vous et pourquoi? (0.5 point)**

En comparant ces deux modes, on aperçoit que la même chose (question 1 de cette section) se produisent. En effet, les 12 Koctets de données sont envoyées dans un seul paquet pour le mode 3 envoie et 300 paquets de 40 bytes pour le mode 4. Selon nous, le mode le plus performante est celui qui envoie une grande quantité de données dans un paquet, donc le mode 3.

3) **Discutez de la fiabilité de chaque mode. Selon vous, quel(s) mode(s) est le plus fiable? (0.5 point)**

Tout d'abord, comparons les mode 1,2 et 3,4. Comme mentionné auparavant, les modes 1 et 2 utilisent le protocole TCP où il y a une synchronisation au début de la communication et les paquets sont vérifiées. Les modes 3 et 4, quant à eux, utilisent le protocole UDP où il n'y a pas de synchronisation ni de vérification de paquets. Ce qui veux dire, que si un paquet est manquant, il n'y aurait pas d'indication et il n'y aura pas de manière de redemander le paquet non plus. Pour cette raison, on conclut que les modes 1 et 2 sont plus fiables que les modes 3 et 4.

Ensuite, comparons les modes 1 et 2. Comme expliqué à la question 1 de cette section, le mode 2 nécessite un reçu de réception de la part du serveur après la transmission de chaque paquet. En d'autres mots, ce mode divise la totalité des données à transmettre en multiples petites paquets nécessitant chacun un reçu du serveur. Alors, en cas d'erreur de réception, on s'aurait rapidement le paquet a la source de l'erreur. On conclut alors, que le mode 2 est le plus fiable et le mode 1 en deuxième place.

4) **Pour les modes secrets utilisant le protocole TCP, vous avez certainement remarqué à la fin de la communication un échange FIN, ACK. Expliquez en quoi consiste cet échange. (0.5 point)**

L'échange FIN, ACK est la confirmation de terminaison de connexion (ACK) et l'information de fin de connexion de la part du récepteur (FIN). Effectivement, le protocole TCP imite une conversation entre 2 personnes qui parlent face à face. À la fin d'une conversation, une personne va dire « Aurevoir! » pour signifier que la conversation est terminée et avant le partir, il s'assure que l'autre personne lui donne un signe de tête ou lui réponds avant de partir.