



École Polytechnique de Montréal
Département de génie informatique et génie logiciel

INF3405

Réseaux informatiques

Automne 2018

Travail pratique #3

Analyse d'applications client-serveur avec Wireshark

Soumis par :

Son-Thang Pham (1856338)

Gabriel Côté-Jones (1771119)

Soumis à :

Dion-Paquin Émilie

Section (04)

5 décembre 2018

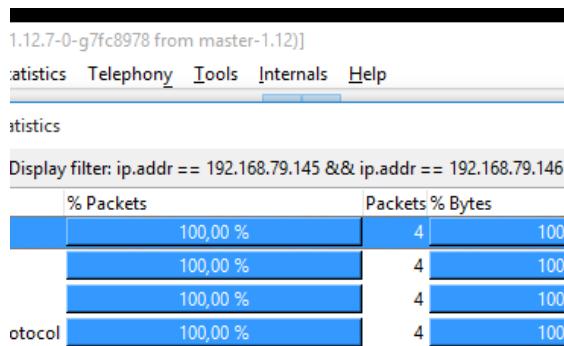
7. Analyse de l'application client/serveur du laboratoire 1 (10 points)

Analyse du flot de données de l'application de gestion de fichier

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



Il est également possible d'utiliser le filtre : ip.addr == (addr ip client) && ip.addr == (addr ip serveur).



- 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. **(0.5 point)**

Protocole TCP.

Filter: `tcp.stream eq 0`

No.	Time	Source	Destination	Protocol	Length	Info
8	9.11827100	192.168.79.145	192.168.79.145	TCP	66	5000-49773 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
9	9.11827100	192.168.79.145	192.168.79.145	TCP	66	5000-49773 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
10	9.11855300	192.168.79.145	192.168.79.145	TCP	60	49773-5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0
11	9.12009200	192.168.79.145	192.168.79.145	TCP	81	5000-49773 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=27
12	9.18121400	192.168.79.145	192.168.79.145	TCP	60	49773-5000 [ACK] Seq=1 Ack=28 Win=65536 Len=0
13	9.18126600	192.168.79.145	192.168.79.145	TCP	111	5000-49773 [PSH, ACK] Seq=28 Ack=1 Win=65536 Len=57
14	9.24365600	192.168.79.145	192.168.79.145	TCP	60	49773-5000 [ACK] Seq=1 Ack=85 Win=65536 Len=0
21	22.9505020	192.168.79.145	192.168.79.145	TCP	76	49773-5000 [PSH, ACK] Seq=1 Ack=85 Win=65536 Len=22
22	23.0015660	192.168.79.145	192.168.79.145	TCP	54	5000-49773 [ACK] Seq=85 Ack=23 Win=65536 Len=0
23	23.0018310	192.168.79.145	192.168.79.145	TCP	69	49773-5000 [PSH, ACK] Seq=23 Ack=85 Win=65536 Len=15
24	23.0019280	192.168.79.145	192.168.79.145	TCP	59	5000-49773 [PSH, ACK] Seq=85 Ack=38 Win=65536 Len=5
25	23.0023850	192.168.79.145	192.168.79.145	TCP	60	49773-5000 [PSH, ACK] Seq=38 Ack=90 Win=65536 Len=6
26	23.0043400	192.168.79.145	192.168.79.145	TCP	1514	49773-5000 [ACK] Seq=44 Ack=90 Win=65536 Len=1460
27	23.0043410	192.168.79.145	192.168.79.145	TCP	1514	49773-5000 [ACK] Seq=1504 Ack=90 Win=65536 Len=1460
28	23.0043410	192.168.79.145	192.168.79.145	TCP	1514	49773-5000 [ACK] Seq=2964 Ack=90 Win=65536 Len=1460
29	23.0043520	192.168.79.145	192.168.79.145	TCP	1476	49773-5000 [PSH, ACK] Seq=4424 Ack=90 Win=65536 Len=1422
30	23.0043830	192.168.79.145	192.168.79.145	TCP	54	5000-49773 [ACK] Seq=90 Ack=5846 Win=65536 Len=0
282	39.0280570	192.168.79.145	192.168.79.145	TCP	78	49773-5000 [PSH, ACK] Seq=5846 Ack=90 Win=65536 Len=24
283	39.0288250	192.168.79.145	192.168.79.145	TCP	57	5000-49773 [PSH, ACK] Seq=90 Ack=5870 Win=65536 Len=3

Frame 9: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: VMware_41:a6:90 (00:0c:29:41:a6:90), Dst: VMware_ff:a1:24 (00:0c:29:ff:a1:24)

Internet Protocol Version 4, Src: 192.168.79.146 (192.168.79.146), Dst: 192.168.79.145 (192.168.79.145)

Transmission Control Protocol, Src Port: 5000 (5000), Dst Port: 49773 (49773), Seq: 0, Ack: 1, Len: 0

0000 00 0c 29 ff a1 24 00 0c 29 41 a6 90 08 00 45 00 ..\$.)A...E.
0010 00 34 5d ed 40 00 80 06 00 00 c0 a8 4f 92 c0 a8 .41@...:O..
0020 4f 91 13 88 c2 6d 0c db f1 3a 28 6e ce 28 80 12 0....m. :.(n(.
0030 20 00 20 9b 00 00 02 04 05 b4 01 03 03 08 01 01
0040 04 02 ..

Ethernet0: <live capture in progress> File: C:\... Packets: 17844 · Displayed: 32 (0,2%) Profile: Default

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

On ajoute au filtre (`ip.dst == addr client`) ou (`ip.dst == addr serveur`) pour filtrer seulement les paquets de la conversation qui vont vers la destination désirée.

Vers le client : 13 paquets

Capturing from Ethernet0 [Wireshark 1.12.7 (v1.12.7-0-g7fc8978 from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: `tcp.stream eq 0 && ip.dst == 192.168.79.145`

No.	Time	Source	Destination	Protocol	Length	Info
9	9.11827100	192.168.79.146	192.168.79.145	TCP	66	5000-49773 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
11	9.12009200	192.168.79.146	192.168.79.145	TCP	81	5000-49773 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=27
13	9.18126600	192.168.79.146	192.168.79.145	TCP	111	5000-49773 [PSH, ACK] Seq=28 Ack=1 Win=65536 Len=57
22	23.0015660	192.168.79.145	192.168.79.145	TCP	54	5000-49773 [ACK] Seq=85 Ack=23 Win=65536 Len=0
24	23.0019280	192.168.79.146	192.168.79.145	TCP	59	5000-49773 [PSH, ACK] Seq=85 Ack=38 Win=65536 Len=5
30	23.0043830	192.168.79.146	192.168.79.145	TCP	54	5000-49773 [ACK] Seq=90 Ack=5846 Win=65536 Len=0
283	39.0288250	192.168.79.145	192.168.79.145	TCP	57	5000-49773 [PSH, ACK] Seq=90 Ack=5870 Win=65536 Len=3
285	39.0879630	192.168.79.146	192.168.79.145	TCP	85	5000-49773 [PSH, ACK] Seq=93 Ack=5870 Win=65536 Len=31
287	39.0908170	192.168.79.146	192.168.79.145	TCP	61	5000-49773 [PSH, ACK] Seq=124 Ack=5875 Win=65536 Len=7
288	39.0913420	192.168.79.146	192.168.79.145	TCP	5894	5000-49773 [ACK] Seq=131 Ack=5875 Win=65536 Len=5840
290	39.0913420	192.168.79.146	192.168.79.145	TCP	5818	5000-49773 [PSH, ACK] Seq=5971 Ack=5875 Win=65536 Len=5764
12556	1306.51509	192.168.79.145	192.168.79.145	TCP	57	5000-49773 [PSH, ACK] Seq=11735 Ack=5879 Win=65536 Len=3
12558	1306.59083	192.168.79.146	192.168.79.145	TCP	91	5000-49773 [PSH, ACK] Seq=11738 Ack=5879 Win=65536 Len=3

Frame 9: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: VMware_41:a6:90 (00:0c:29:41:a6:90), Dst: VMware_ff:a1:24 (00:0c:29:ff:a1:24)

Internet Protocol Version 4, src: 192.168.79.146 (192.168.79.146), Dst: 192.168.79.145 (192.168.79.145)

Transmission Control Protocol, Src Port: 5000 (5000), Dst Port: 49773 (49773), Seq: 0, Ack: 1, Len: 0

0000 00 0c 29 ff a1 24 00 0c 29 41 a6 90 08 00 45 00 ..\$.)A...E.
0010 00 34 5d ed 40 00 80 06 00 00 c0 a8 4f 92 c0 a8 .41@...:O..
0020 4f 91 13 88 c2 6d 0c db f1 3a 28 6e ce 28 80 12 0....m. :.(n(.
0030 20 00 20 9b 00 00 02 04 05 b4 01 03 03 08 01 01
0040 04 02 ..

Ethernet0: <live capture in progress> File: C:\... Packets: 16981 · Displayed: 13 (0,1%) Profile: Default

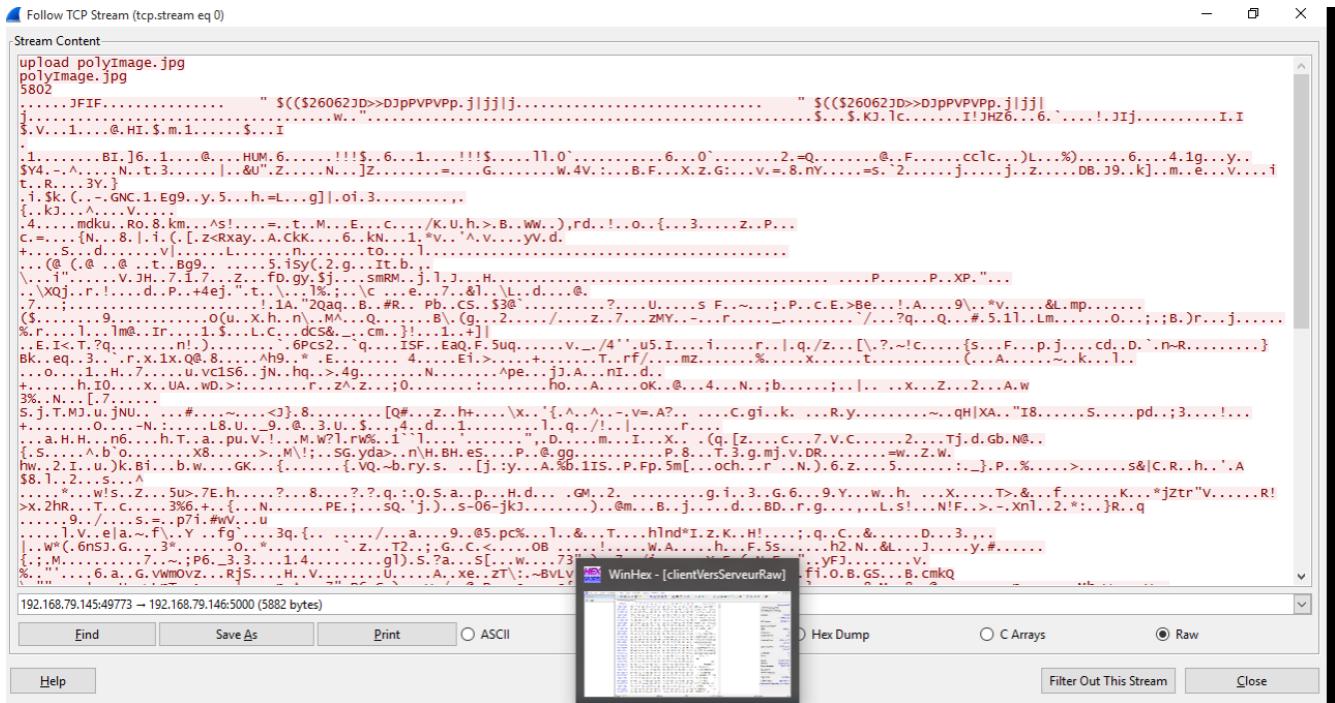
Vers le serveur : 19 paquets

Wireshark capture showing 19 TCP packets to port 79.145. The details pane highlights the first few packets, and the bytes pane shows the raw hex and ASCII data.

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 on retrouve des paquets de taille au-dessus de 5000 octets.

Wireshark capture showing a single TCP packet (Frame 288) with a length of 5894 bytes. The details pane highlights the packet, and the bytes pane shows the raw data.

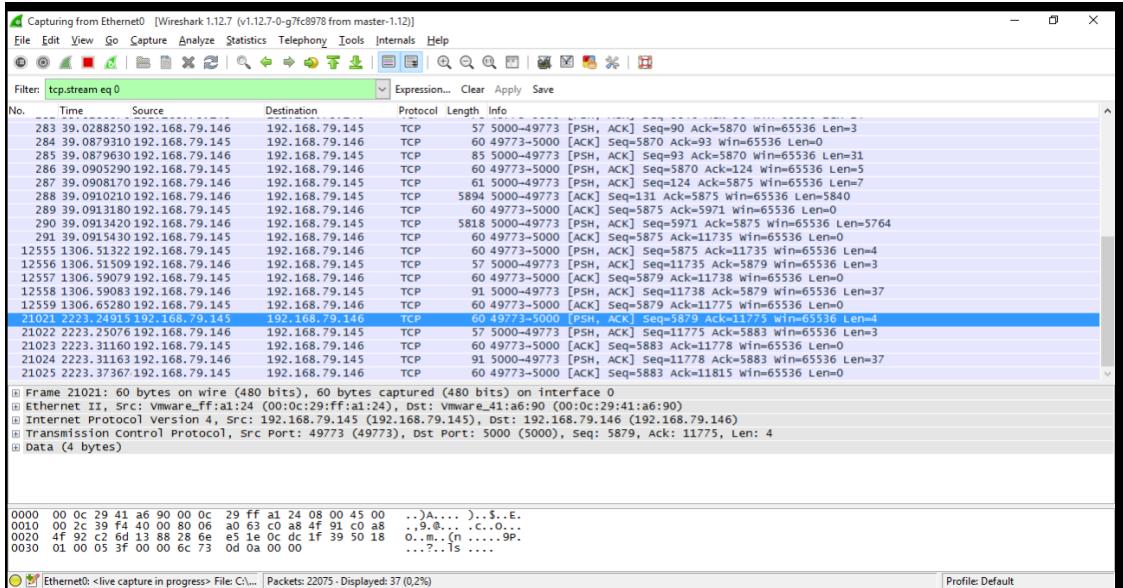


On voit dans la deuxième image que le paquet est celui de téléversement de l'image polyImage.jpg. Comme il n'est pas possible de dépasser la limite imposée (à moins de l'augmenter manuellement), il est fort probable que ces paquets représentent la reconstitution (après transmission) des différents paquets qui formaient l'image.

Après un peu de recherche, il semble qu'il y a une option TCP nommée « large segment offload » pour faire passer des plus gros paquets à la carte réseau, qui elle les brise en paquets de taille respectant la limite avant de les transmettre. La même option pour la réception est également possible (la carte réseau reconstitue avant de passer à l' OS). Comme cela se fait après l' OS de la source et avant l' OS du destinataire, Wireshark capte les paquets comme un seul paquet de taille au-dessus de la limite.

5) Quel type d'information êtes-vous capables d'extraire de Wireshark en lien l'exécution de la commande « ls »? Montrer vos résultats (**1 point**)

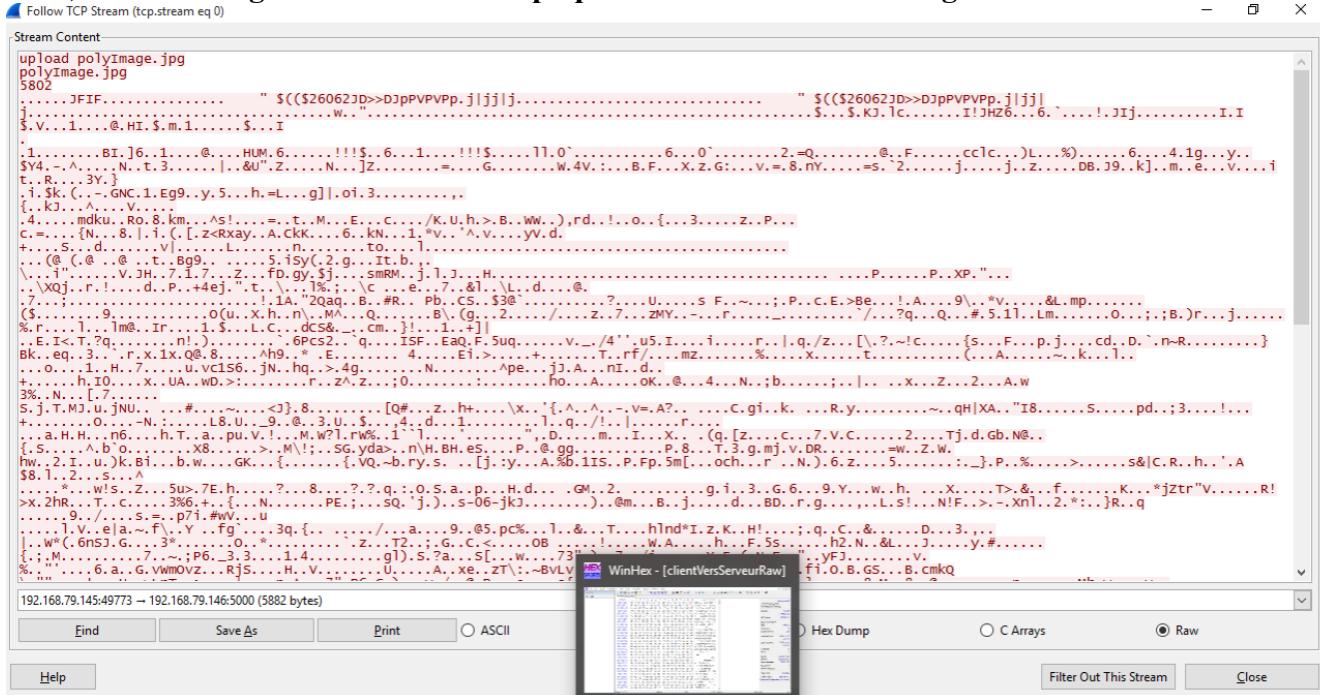
Les paquets du ls commencent à paquet #21021 dans l'image ci-dessous. À partir du paquet, on peut obtenir des informations comme la taille du paquet de la commande ls (60 octets) ou la taille des données du paquet (4 octets). On voit aussi que le client indique au serveur de ne pas attendre plus d'information avant d'envoyer la réponse (PSH).



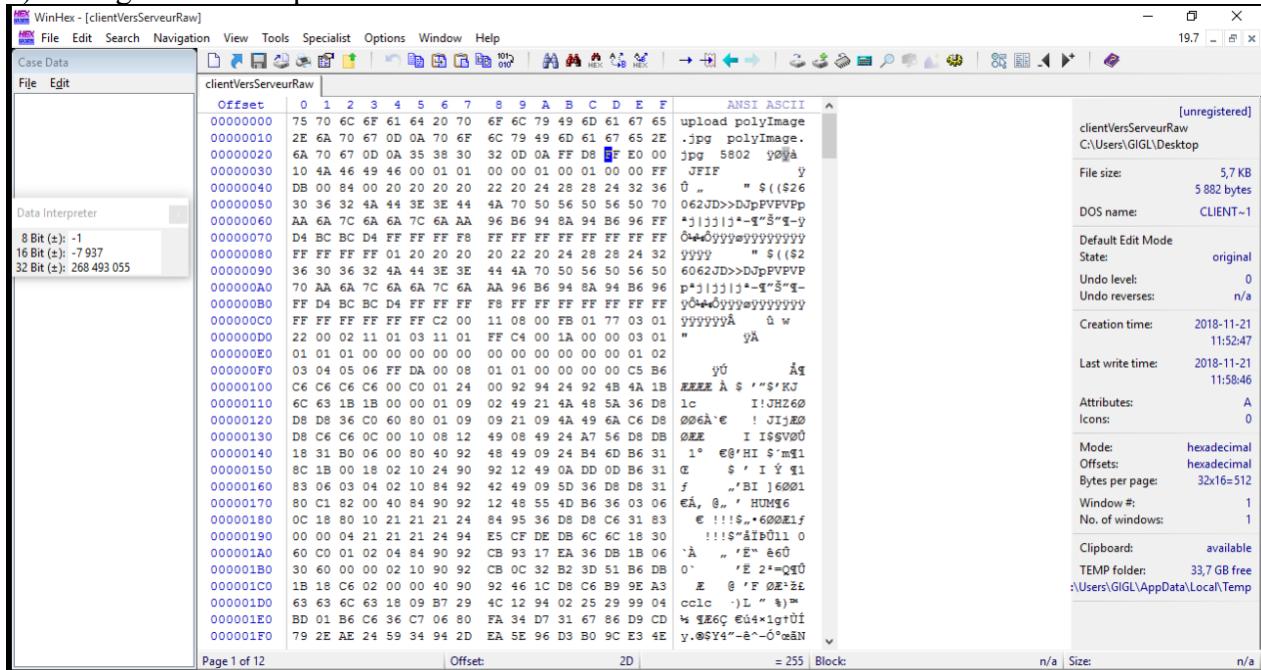
6) Il est possible, avec Wireshark, d'extraire l'image envoyée par le client ou l'image envoyée par le serveur vers le client. Donnez les étapes à suivre, incluant des captures d'écran montrant chaque étape permettant l'extraction de l'image envoyée du client vers le serveur. Servez-vous des propriétés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

- On filtre la conversation pour ne laisser que client vers serveur.
- On sauvegarde le fichier en raw et on l'ouvre dans winHex.
- On enlève le bloc de données avant FF D8 FF E0 et après FF D9
- On enregistre le résultat avec l'extension .jpg

a) Voici l'image du fichier raw du paquet téléversement de l'image en format raw.

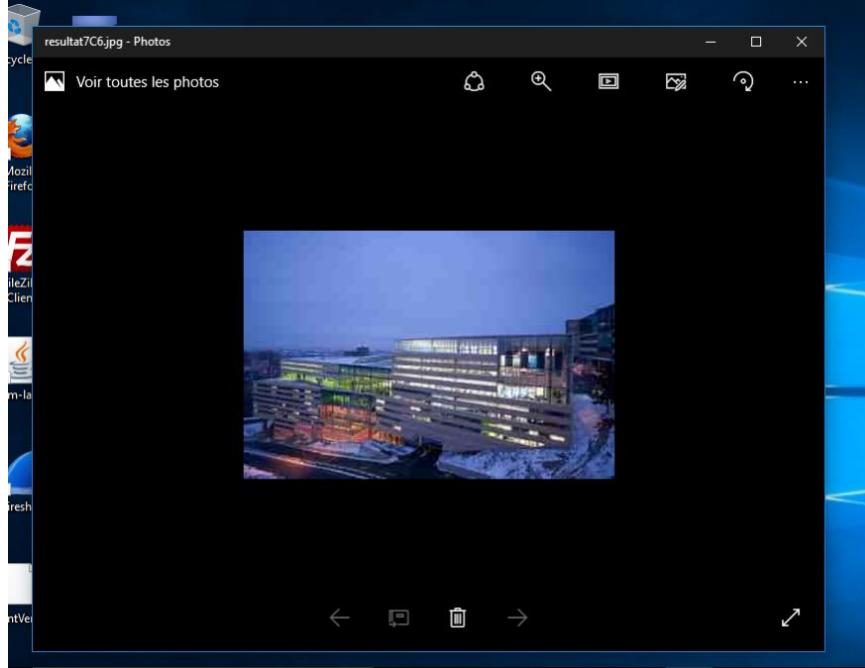


b) L'image suivante représente le fichier raw ouvert avec WinHex.



c) Les deux images suivantes représentent le fichier dans WinHex après avoir supprimé les données avant FF D8 FF E0 et après FF D9

d) L'image suivante est celle de du fichier raw de l'étape c sauvegardée avec l'extension .jpg.



7) À la suite de toute cette analyse que pouvez-vous conclure quant à la sécurité de l'application de gestion de fichier que vous avez développé lors du travail pratique no.1 (**1 point**)

Il semble que même si on intercepte les paquets TCP de l'échange client/serveur, il faut un moyen de le déchiffrer pour en soutirer l'information. Comme nous n'avons pas implémenté quoique ce soit pour empêcher quelqu'un d'intercepter les échanges et qu'il semble y avoir des normes pour reconnaître et déchiffrer l'information, le programme développer lors du T.P.1 n'est pas très sûre.

Verdict : À éviter pour faire passer de l'information sensible...

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

Mode secret 1

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 a lieu? Pourquoi? (**0.5 point**)

TCP. C'est l'échange de synchronisation pour établir une connexion. Le premier message du client au serveur est la demande de connexion SYN (image 1). La réponse du serveur (image 2) SYN ACK est la confirmation du serveur. Le deuxième paquet envoyé par le client (image 1) est la confirmation du client (ACK).

Si c'était UDP, ce premier échange n'aurait pas lieu, car UDP est un protocole sans connexion.

Image 1 :

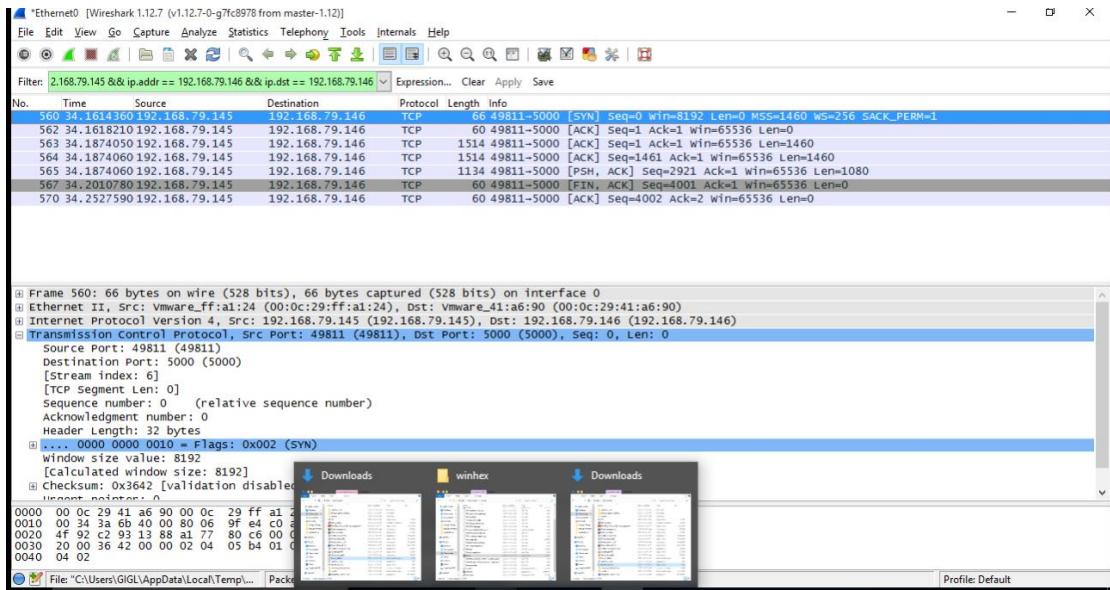
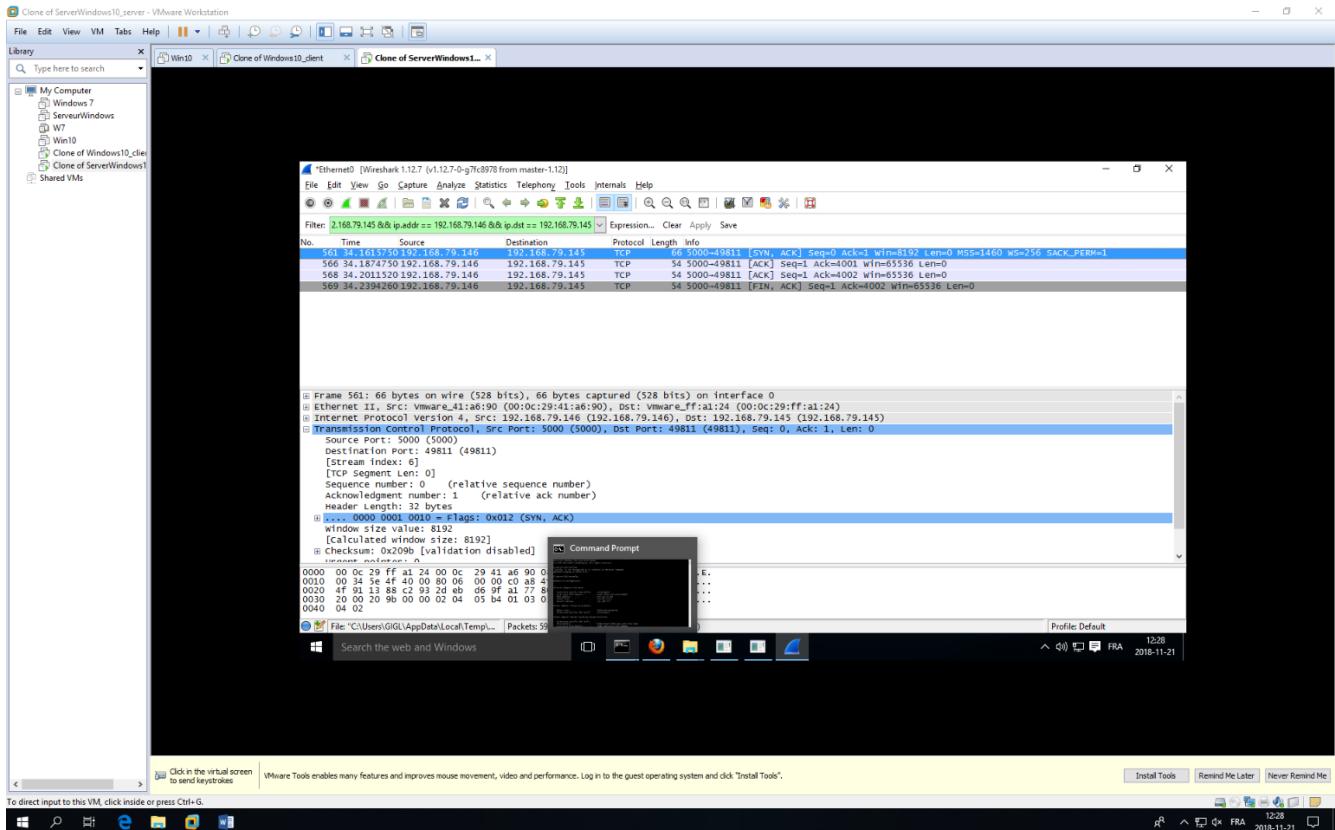
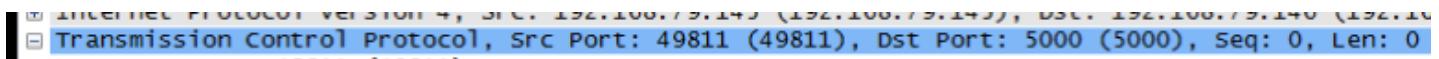


Image 2 :



2) En vous basant sur les informations recueillies par Wireshark, indiquez les ports source et destinations utilisées par la couche 4. (**0.5 point**)

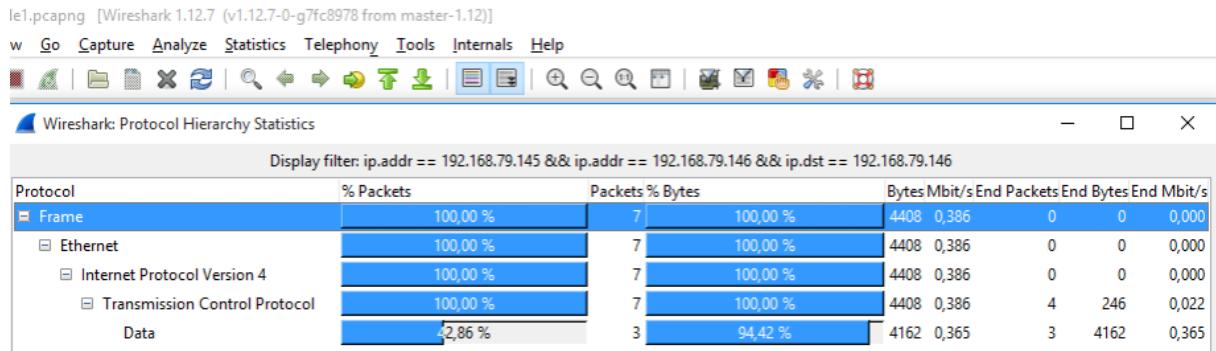
5000 du côté serveur (dest) et 49811 du côté client (src). L'image plus haut montre que la destination était le IP du serveur.



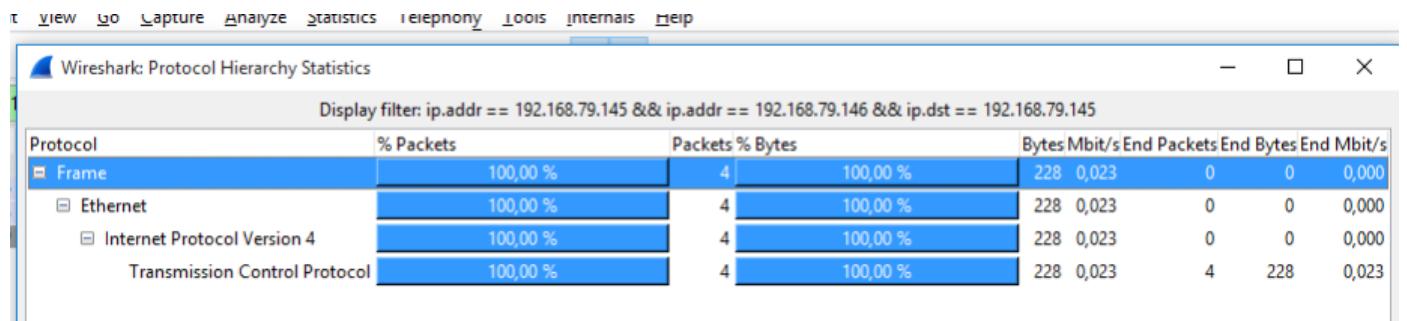
3) Combien de paquets et d'octets contenant des données ont été envoyés par le client vers le serveur? Par le serveur vers le client? Montrer où vous avez trouvé cette information. (**0.5 point**)

On utilise un filtre pour avoir seulement des paquets de la conversation désirée se dirigeant vers la destination désirée (client ou serveur). On peut ensuite ouvrir l'onglet « Protocol Hierarchy Statistics » pour obtenir le nombre de paquets et d'octets de données total selon le filtre appliqué.

Client vers serveur : 7 paquets et 4162 octets de données.



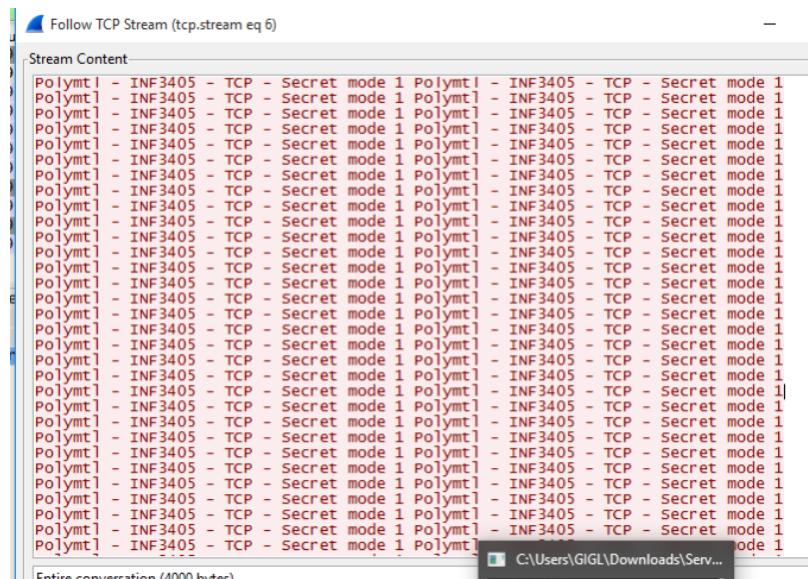
Serveur vers client : 4 paquets et 0 octet de données.



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)

Le client envoie la même ligne plusieurs fois. Il le fait en 6 itérations.

On obtient le nombre d'itérations en observant l'échange sous forme de tableau.



Follow TCP Stream (tcp.stream eq 6)

Stream Content

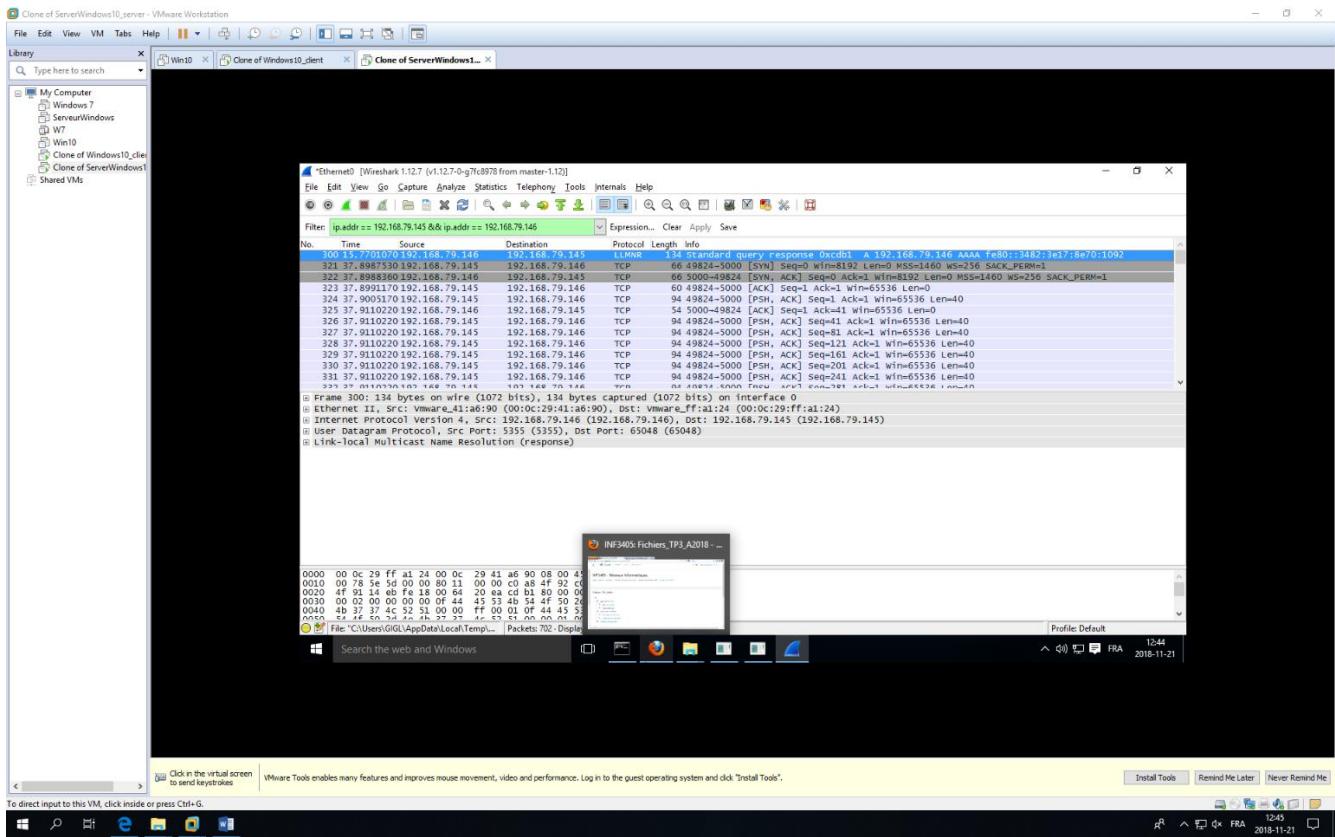
```
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x20,
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x20,
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x20,
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x20,
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20 },
char peer0_5[] = {
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x20,
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x31, 0x20 }
```

Mode secret 2

- 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? **(0.5 point)**

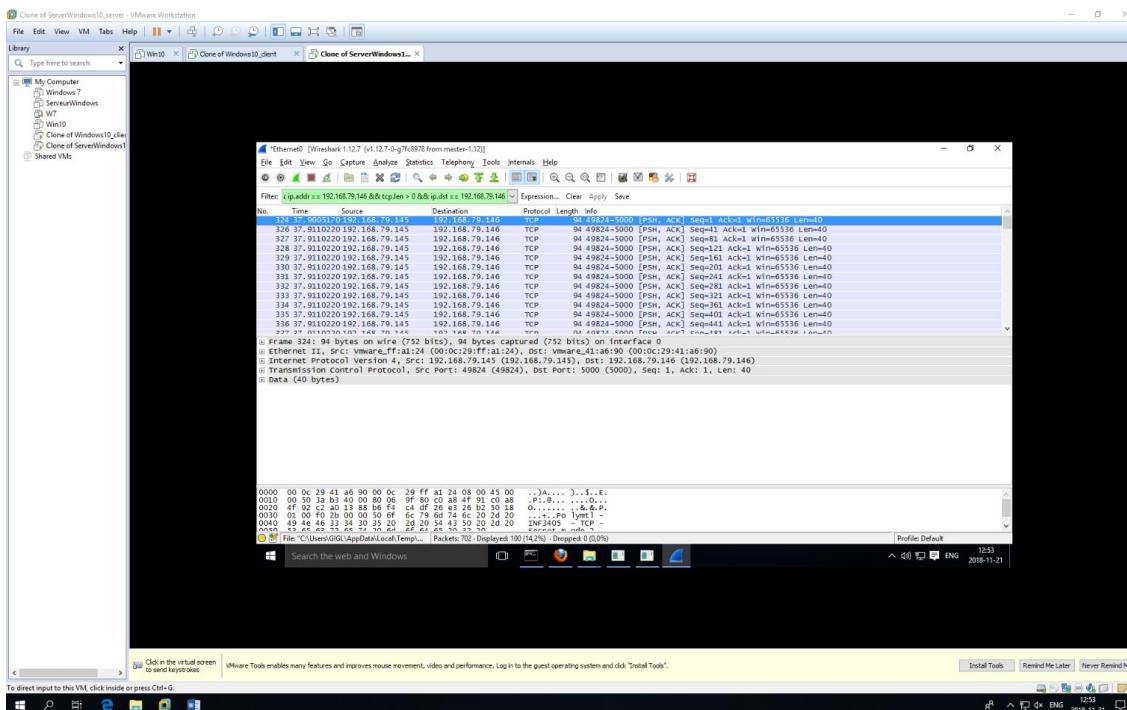
TCP. C'est l'échange de synchronisation pour établir une connexion. Le premier message du client au serveur est la demande de connexion SYN. La réponse du serveur SYN ACK est la confirmation du serveur. Le deuxième paquet envoyé par le client est la confirmation du client (ACK).

Si c'était UDP, ce premier échange n'aurait pas lieu, car UDP est un protocole sans connexion.



2) En vous basant sur les informations recueillies par Wireshark, indiquez les ports source et destinations utilisées par la couche 4. (**0.5 point**)

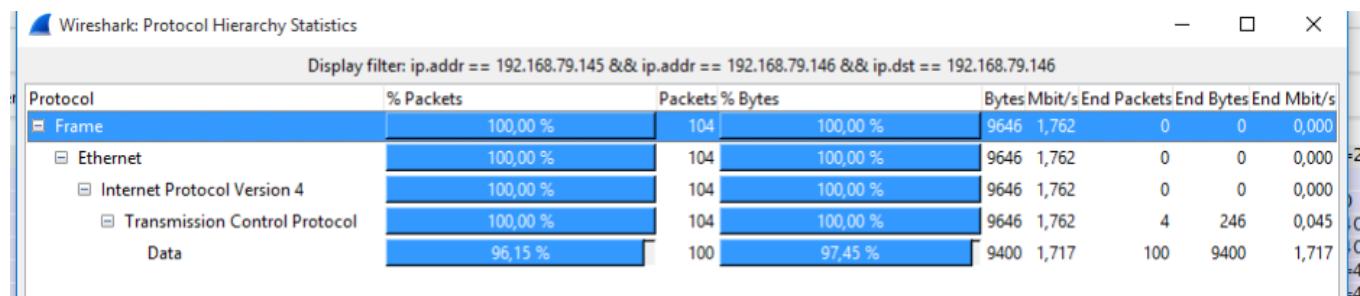
49824 du côté client (src) et 5000 du côté serveur (dest).



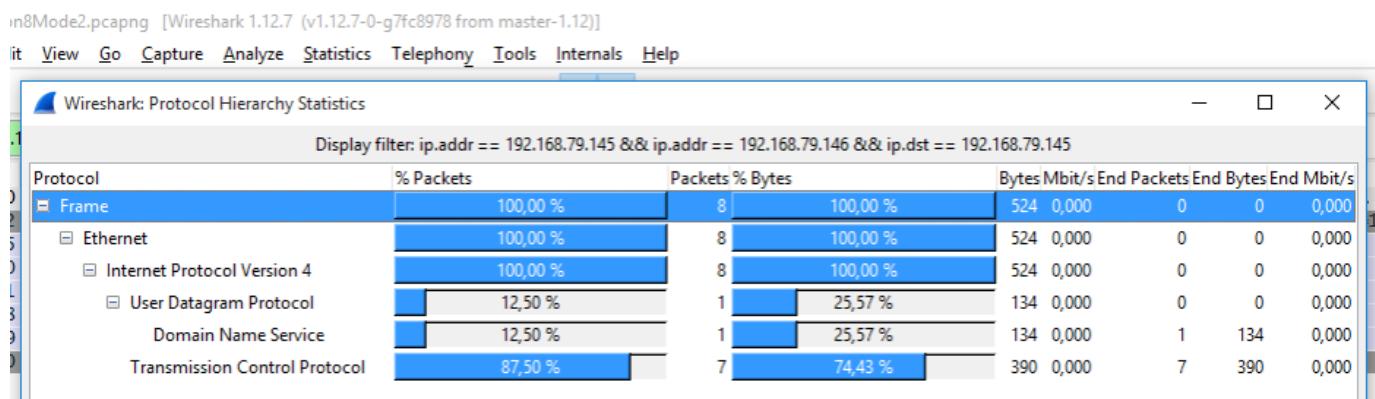
3) Combien de paquets et d'octets contenant des données ont été envoyés par le client vers le serveur? Par le serveur vers le client? Montrer où vous avez trouvé cette information. (**0,5 point**)

On utilise un filtre pour avoir seulement des paquets de la conversation désirée se dirigeant vers la destination désirée (client ou serveur). On peut ensuite ouvrir l'onglet « Protocol Hierarchy Statistics » pour obtenir le nombre de paquets et d'octets de données total selon le filtre appliqué.

Client vers serveur : 104 paquets et 9400 octets de données.



Serveur vers client : 8 paquets et 0 octet de données.



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**)

Même chose que le mode 1, c'est-à-dire envoie une même ligne de texte à répétition, mais en 100 itérations.

Follow TCP Stream (tcp.stream eq 6)

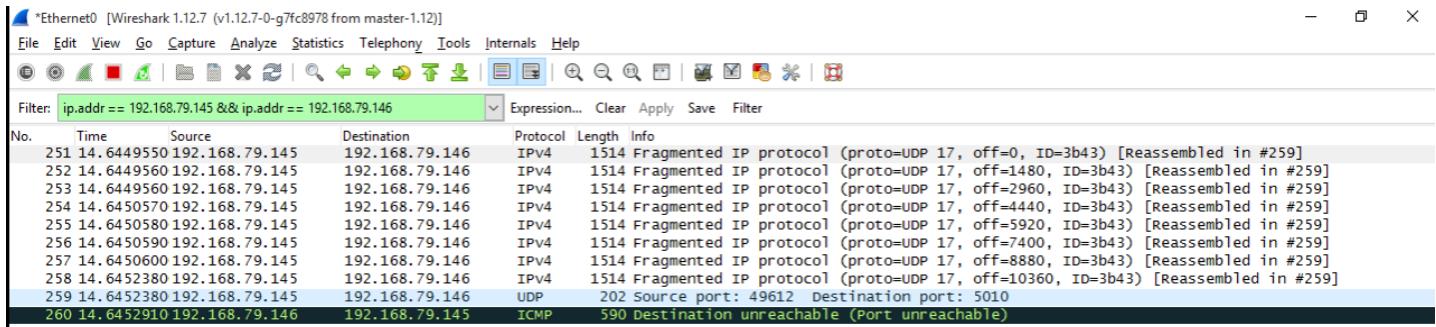
Stream Content

```
char peer0_95[] = {  
    0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,  
    0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,  
    0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,  
    0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,  
    0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x32, 0x20 };  
char peer0_96[] = {  
    0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,  
    0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,  
    0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,  
    0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,  
    0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x32, 0x20 };  
char peer0_97[] = {  
    0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,  
    0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,  
    0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,  
    0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,  
    0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x32, 0x20 };  
char peer0_98[] = {  
    0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,  
    0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,  
    0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,  
    0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,  
    0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x32, 0x20 };  
char peer0_99[] = {  
    0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,  
    0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,  
    0x35, 0x20, 0x2d, 0x20, 0x54, 0x43, 0x50, 0x20,  
    0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,  
    0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x32, 0x20 };
```

Mode secret 3

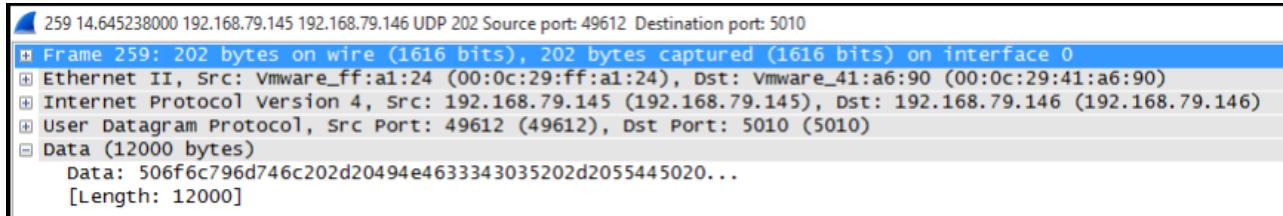
1) Quel protocole de la couche transport est utilisé? Dans le cas de TCP, montrer le tout premier échange entre le client et le serveur lors de l'initialisation de la connexion, comment ce nomme cet échange? Dans le cas d'UDP, est-ce que ce même échange à lieu? Pourquoi? (**0.5 point**)

C'est le protocole UDP. Comme c'est UDP, ce premier échange n'a pas lieu, car UDP est un protocole sans connexion.



2) En vous basant sur les informations recueillies par Wireshark, indiquez les ports source et destinations utilisées par la couche 4. (**0.5 point**)

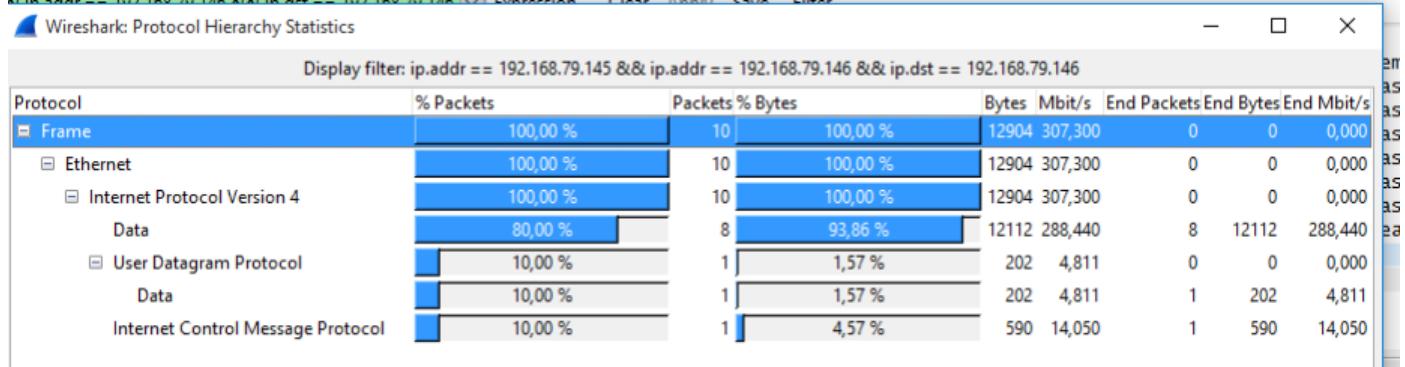
49612 pour le client (src) et 5010 pour le serveur (dest).



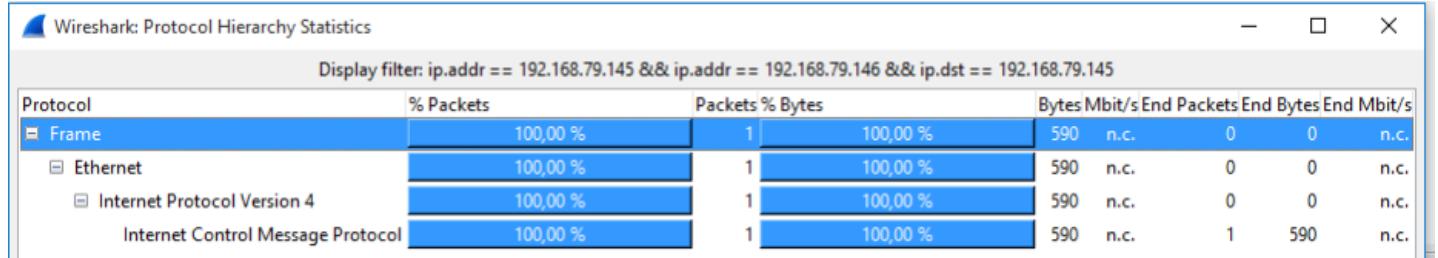
3) Combien de paquets et d'octets contenant des données ont été envoyés par le client vers le serveur? Par le serveur vers le client? Montrer où vous avez trouvé cette information. (**0.5 point**)

On utilise un filtre pour avoir seulement des paquets de la conversation désirée se dirigeant vers la destination désirée (client ou serveur). On peut ensuite ouvrir l'onglet « Protocol Hierarchy Statistics » pour obtenir le nombre de paquets et d'octets de données total selon le filtre appliqué.

Client vers serveur : 10 paquets et 12112 + 202 octets de données (un paquet IP et un paquet UDP contenant des données). À noter : seulement les 202 octets représentent les données encapsulées par le protocole UDP.

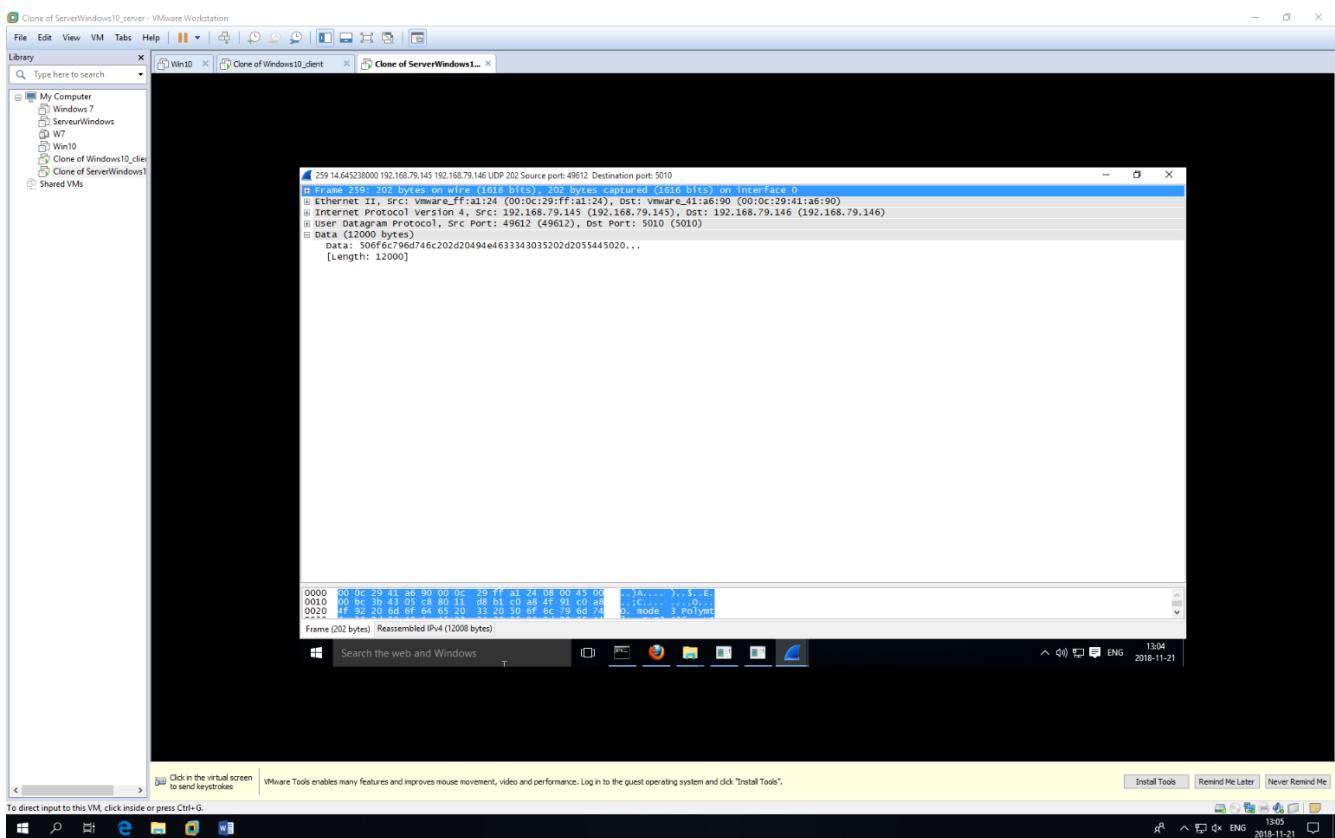


Serveur vers client : 1 paquet et 0 octet de données.



- 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**)

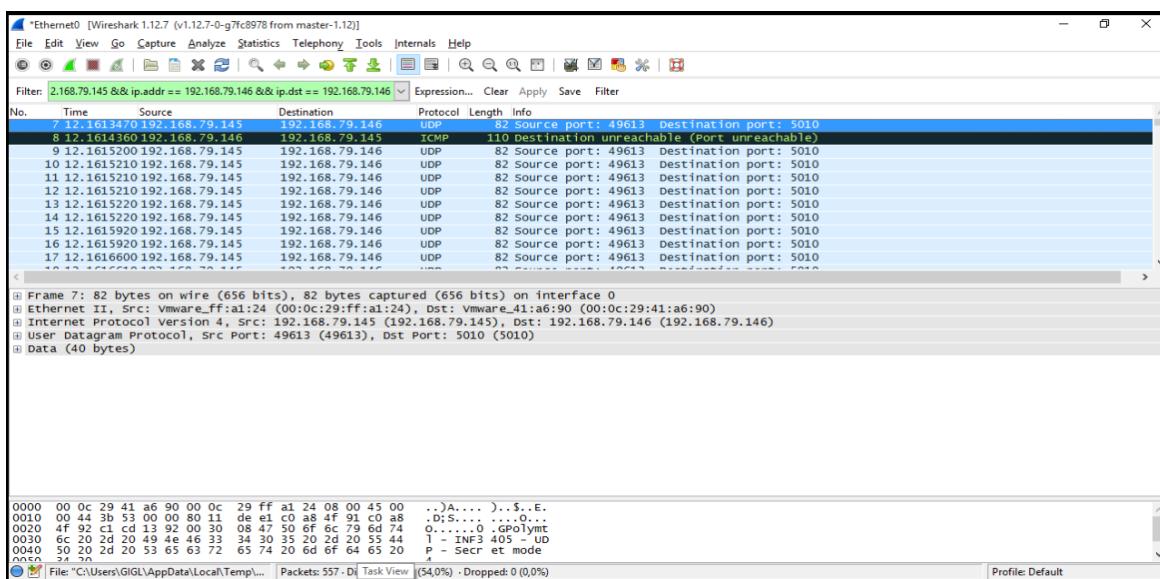
Le client tente d'envoyer un seul paquet UDP avant de recevoir un ICMP l'informant que le numéro de port auquel il envoie n'est pas atteignable.

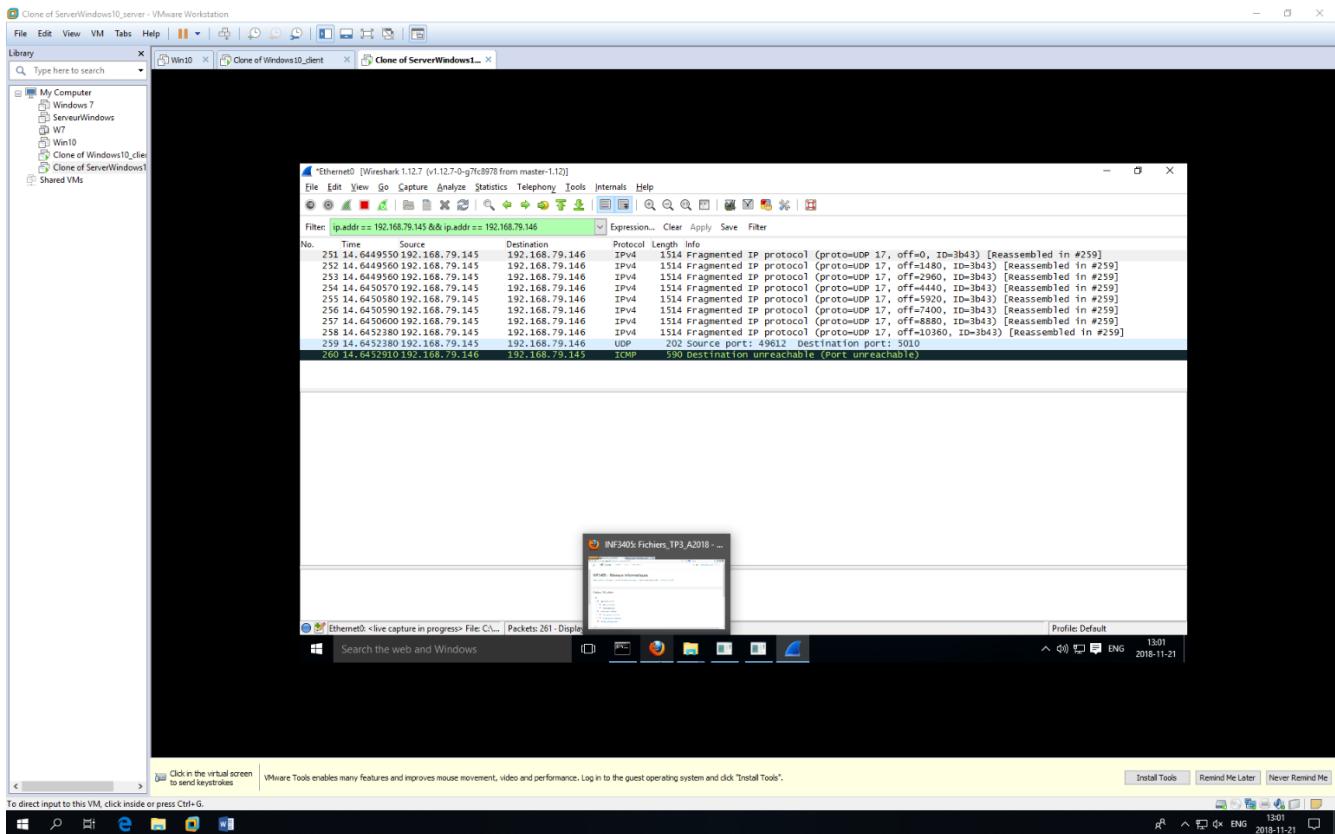


Mode secret 4

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? (**0.5 point**)

C'est le protocole UDP. Comme c'est UDP, ce premier échange n'a pas lieu, car UDP est un protocole sans connexion.





2) En vous basant sur les informations recueillies par Wireshark, indiquez les ports source et destinations utilisées par la couche 4. (**0.5 point**)

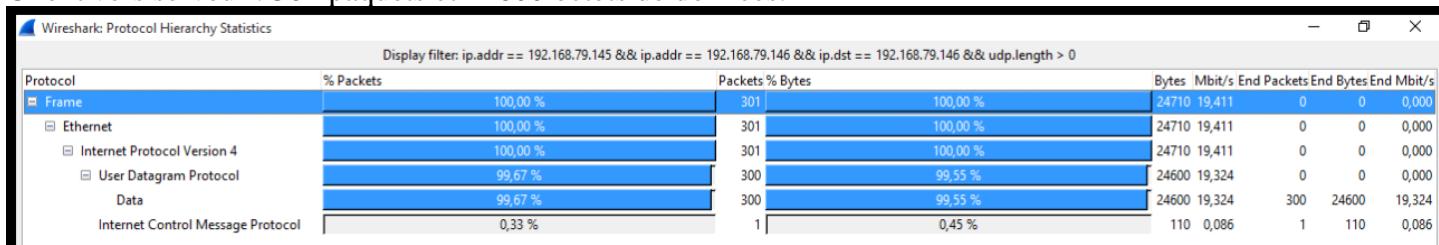
49613 = port client (src) et 5010 = serveur (dest)

No.	Time	Source	Destination	Protocol	Length	Info
7	12.1613470	192.168.79.145	192.168.79.146	UDP	82	Source port: 49613 Destination port: 5010

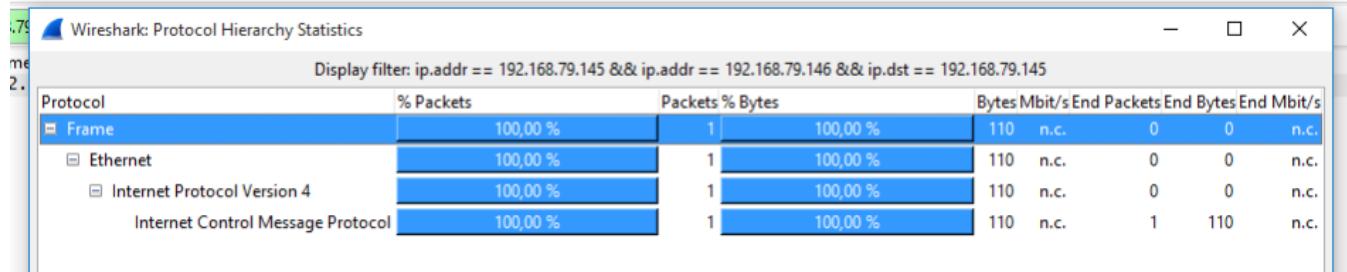
3) Combien de paquets et d'octets contenant des données ont été envoyés par le client vers le serveur? Par le serveur vers le client? Montrer où vous avez trouvé cette information. (**0.5 point**)

On utilise un filtre pour avoir seulement des paquets de la conversation désirée se dirigeant vers la destination désirée (client ou serveur). On peut ensuite ouvrir l'onglet « Protocol Hierarchy Statistics » pour obtenir le nombre de paquets et d'octets de données total selon le filtre appliqué.

Client vers serveur : 301 paquets et 24600 octets de données.

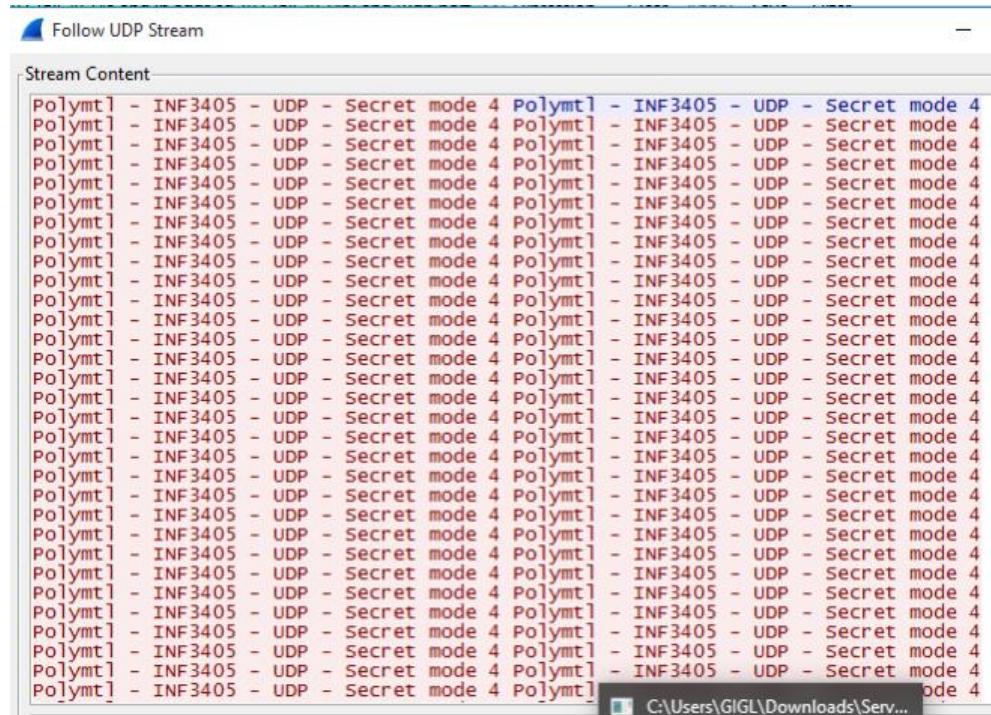


Serveur vers client : 1 paquet et 0 octet de données.



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)

Encore une fois, le client envoie la même ligne de texte à répétition et il le fait en 300 itérations.



Stream Content

```

char peer0_295[] = {
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x55, 0x44, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x34, 0x20 };
char peer0_296[] = {
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x55, 0x44, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x34, 0x20 };
char peer0_297[] = {
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x55, 0x44, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x34, 0x20 };
char peer0_298[] = {
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x55, 0x44, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x34, 0x20 };
char peer0_299[] = {
0x50, 0x6f, 0x6c, 0x79, 0x6d, 0x74, 0x6c, 0x20,
0x2d, 0x20, 0x49, 0x4e, 0x46, 0x33, 0x34, 0x30,
0x35, 0x20, 0x2d, 0x20, 0x55, 0x44, 0x50, 0x20,
0x2d, 0x20, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74,
0x20, 0x6d, 0x6f, 0x64, 0x65, 0x20, 0x34, 0x20 }.
```

 Downloads  winh

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)**

Le mode 1 envoi en moins d'itération. Comme ils utilisent tous les deux le protocole de transport TCP, il y a une vérification que l'information a bien été reçue.

Comme il n'y a pas de crainte que l'information ne soit acheminée qu'à moitié, le mode 1 qui envoie en moins d'itérations est plus performant.

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)**

Le mode 4 est plus performant, car il utilise un numéro de port valable... De plus, il semble que le mode 3 tentait de tout envoyer en un paquet. Bien que plus rapide, si le paquet est perdu, le serveur aura reçu 0% de l'information. Le mode 4 envoi en beaucoup d'itérations, même s'il perd un ou deux paquets, le serveur aura quand même la majorité de l'information qui lui était destinée.

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

Le mode 1 et le mode 2 ont tous deux des confirmations/surveillance de réception de paquet (ACK), car ils utilisent TCP.

Le mode 3 risque de mal acheminer la totalité de l'information et semble ne pas avoir le bon numéro de port.

Le mode 4 envoie l'information, mais semble perdre un ou deux paquets. Comme il utilise le protocole de transport UDP, ces paquets ne seront pas réenvoyés.

Les modes 1 et 2 sont donc les plus fiables.

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)**

C'est l'échange de terminaison de la connexion TCP pour fermer la connexion établie lors du premier échange. Le client envoi un paquet de terminaison FIN reçoit une confirmation ACK du serveur puis envoi à son tour une confirmation ACK.