École Polytechnique de Montréal



INF3405

Réseaux informatiques

Hiver 2020 laboratoire 3

Soumis par

Roman Zhornytskiy (1899786) et

# [Hakim Payman](https://www.facebook.com/hakim.sakhawat) (1938609)

Soumis à Esther Guerrier

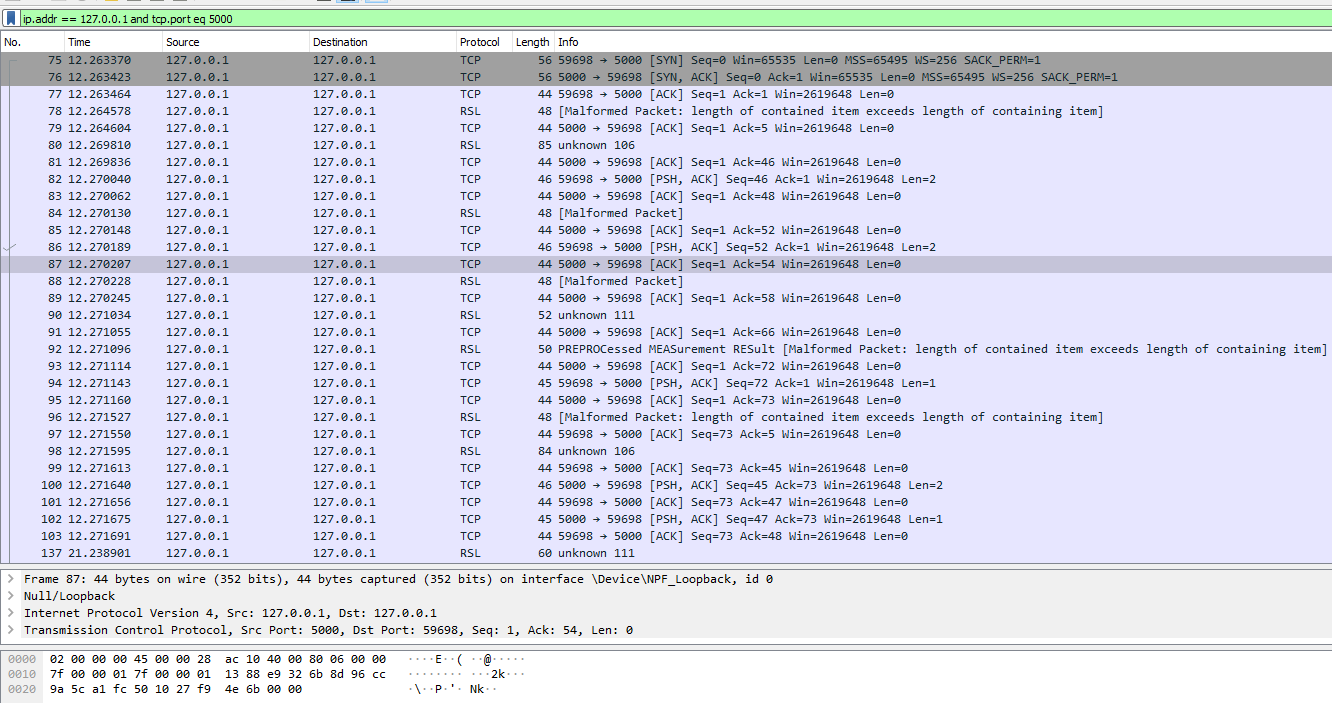
15 avril 2020

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

ip.addr == 127.0.0.1 and tcp.port == 5000. Ce filtre nous permet de visualiser seulement les communications entre le client et le serveur qui sont tous les deux sur l’addresse IP 127.0.0.1. L’ajout est nécessaire afin de ne garder que les échanges qui partent du port du serveur ou qui se dirige vers ce port.

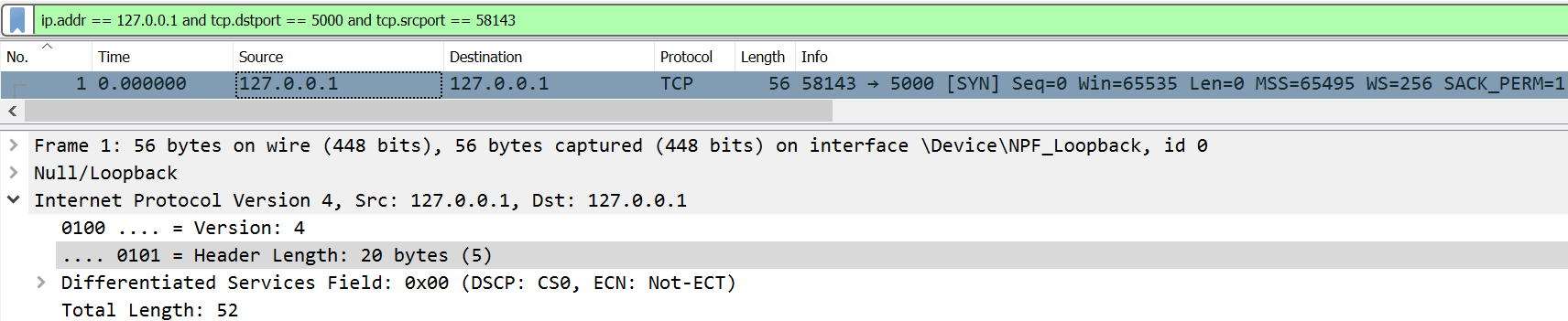
2) À la lumière de vos observations, dites quel protocole de la couche 4 est utilisé pour la communication entre le client et le serveur. **(0.5 point)**

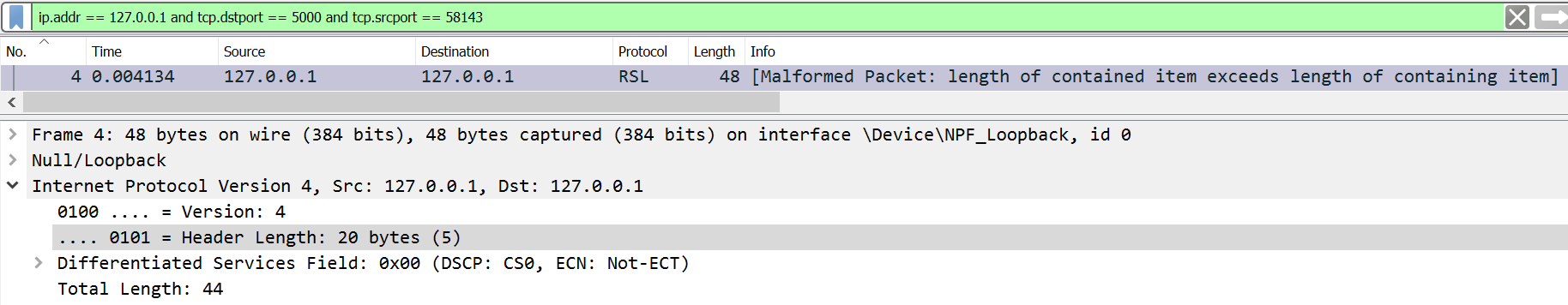
TCP

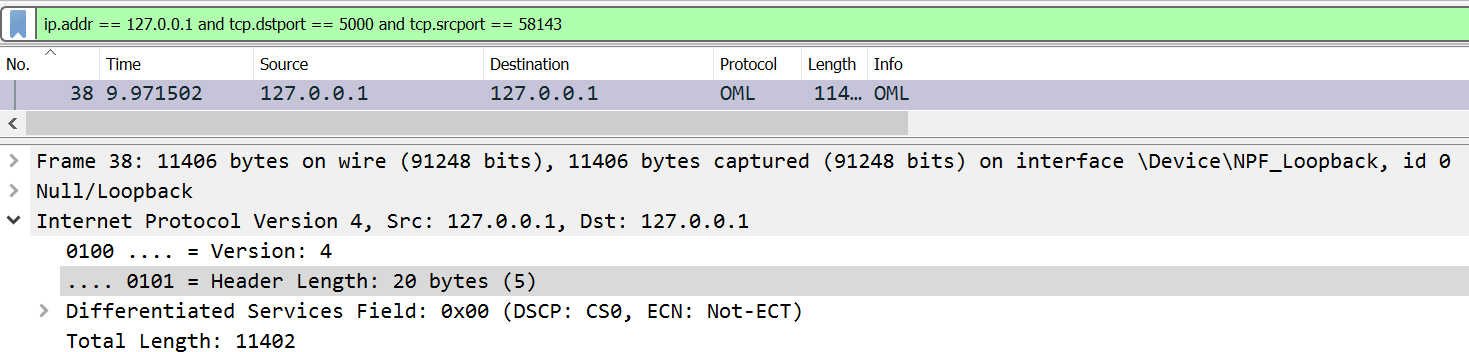


3) Combien de paquets et d’octets de données ont été envoyés du client vers le serveur et du serveur vers le client ? **(2 points)**

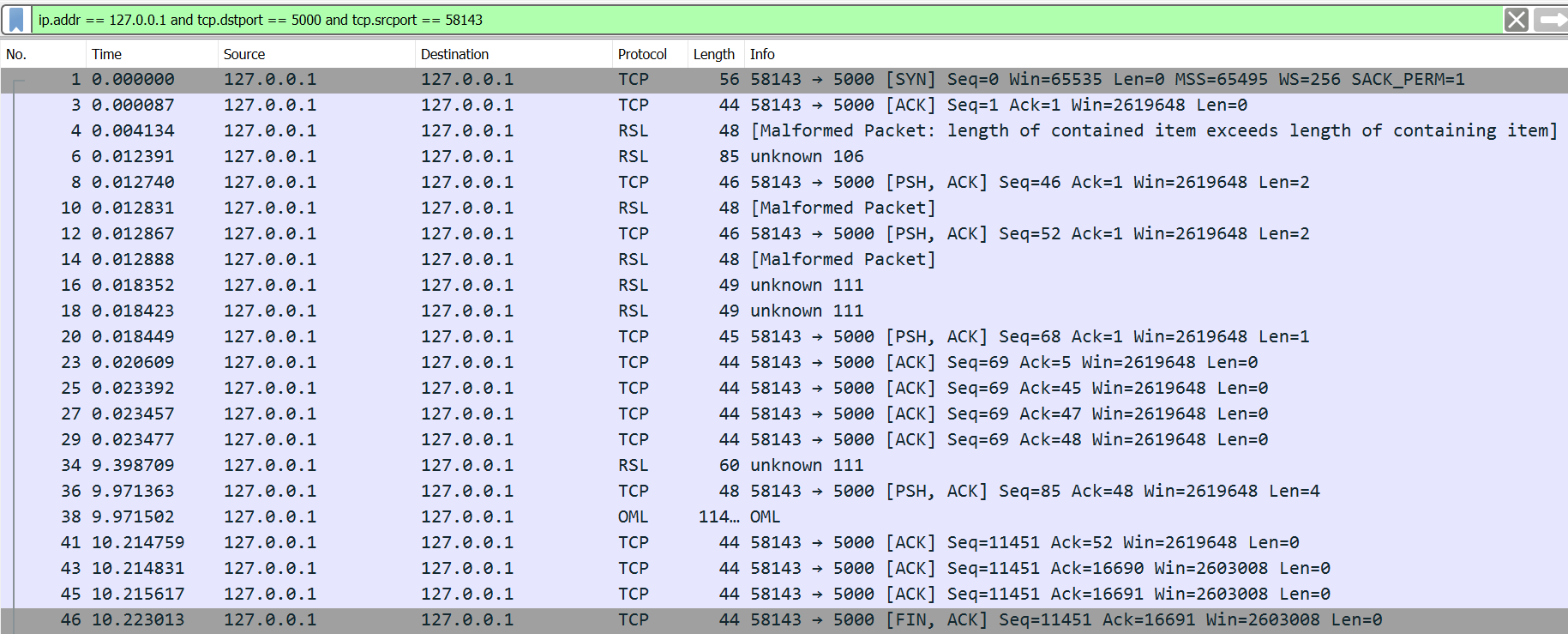
Dans l’ensemble, nous avons vu qu’il n’y a que des paquets TCP, RSL et OML qui sont échangés entre le client et le serveur. En utilisant Wireshark, nous pouvons déterminer la taille des en-têtes de chacun des paquets et ainsi calculer la quantité de données dans chacun des paquets en soustrayant la taille de son en-tête avec la taille totale (length) du paquet. Vu que nous communiquons avec le protocole TCP, les en-têtes ont une longueur de 20 octets. Il est à noter que certains paquets emploient un autre protocole avec le protocole TCP. Ces derniers sont le RSL et le OML (voir les trois figures suivantes).





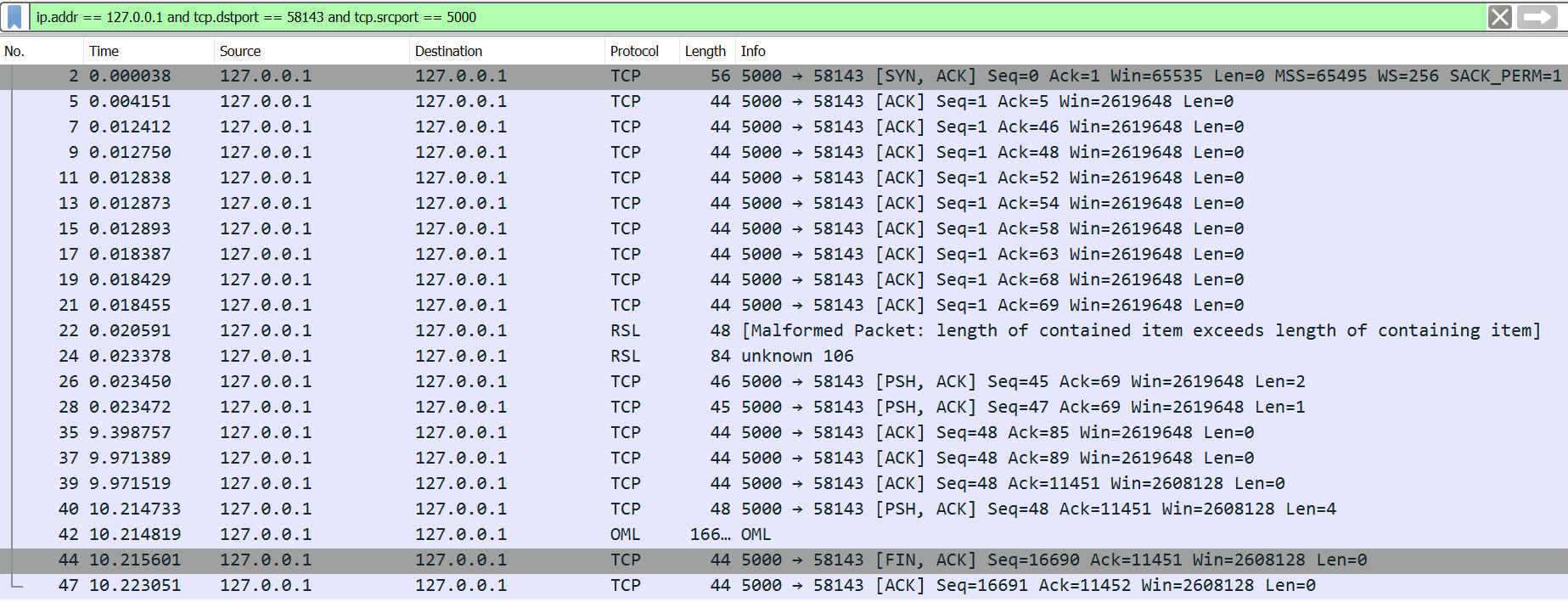


L’image suivante montre les paquets que le client envoit au serveur (nous avons utilisé le filtre « ip.addr == 127.0.0.1 and tcp.dstport == 5000 and tcp.srcport == 58143 » afin d’identifier les paquets en direction du serveur).



Sur cette capture d’écran, nous voyons qu’il y a un total de 22 paquets en provenance du client vers le serveur. Parmi ces paquets, nous trouvons 14 paquets envoyés en utilisant le protocole TCP, 7 paquets envoyés en utilisant le protocole TCP avec RSL et 1 paquet envoyé en utilisant le protocole TCP avec OML. Pour trouver le nombre total d’octets de données envoyé, il suffit d’additionner la taille (length) de tous les paquets et de soustraire le nombre de paquets multiplié par 20 octets (vu que c’est la taille de toutes les en-têtes). Nous obtenons ainsi :

L’image suivante montre les paquets que le serveur envoit au client (nous avons utilisé le filtre « ip.addr == 127.0.0.1 and tcp.dstport == 58143 and tcp.srcport == 5000 » afin d’identifier les paquets en direction du client).



Sur cette capture d’écran, nous voyons qu’il y a un total de 21 paquets en provenance du serveur vers le client. Parmi ces paquets, nous trouvons 18 paquets envoyés en utilisant le protocole TCP, 2 paquets envoyés en utilisant le protocole TCP avec RSL et 1 paquet envoyé en utilisant le protocole TCP avec OML. Le même calcul fait précédemment s’applique ici pour déterminer la quantité d’octets de donné envoyés. Nous obtenons ainsi :

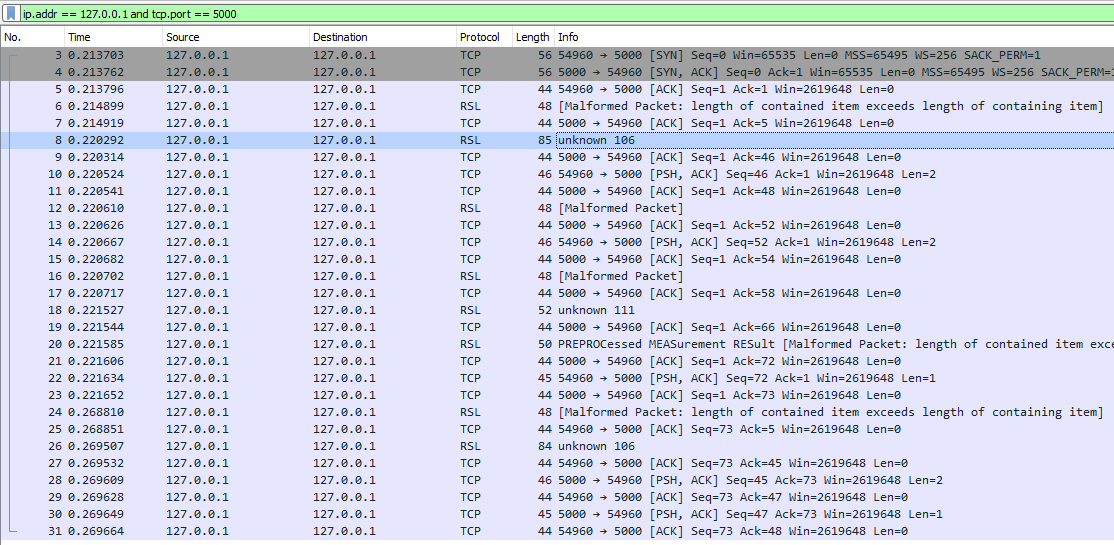
4) Normalement, le standard IEEE 802.3 limite la taille d’une trame *Ethernet* à 1518 octets. Dans votre capture Wireshark, existe-t-il des paquets ayant une taille supérieure à 1518 octets? Si oui, expliquez pourquoi et comment ce paquet réussit à transiger sur le réseau alors que sa taille est plus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

Oui, dans notre cas, nous avons 2 paquets qui ont une taille de 11402 octets et de 16678 octets. Ils correspondent, respectivement, à l’image originale en provenance du client et à l’image traitée en provenance du serveur. Il est possible de faire transiger un paquet de cette taille, car la couche réseau s’occupe de découper ce gros paquet en plus petits paquets qui respectent la taille maximale que le réseau peut supporter. Au moment de la réception par le destinataire, ces paquets sont rassemblés en un paquet dans l’ordre original. Cependant, Wireshark nous donne des informations au niveau de la couche de transport (protocole TCP). Ainsi, nous ne pouvons voir que le paquet une fois rassemblé.

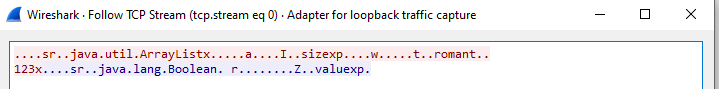
5) Quel type d’information êtes-vous capables d’extraire de Wireshark en lien avec l’authentification au serveur de traitement d’images? **(1 point)**

Avec l’aide de Wireshark on est capable d’extraire le nom d’utilisateur et son mot de passe, car ces données ne sont pas encryptées.

En se connectant au serveur avec le nom d’utilisateur et le mot de passe on obtient une première série de paquets TCP qui est dans la photo en dessous.



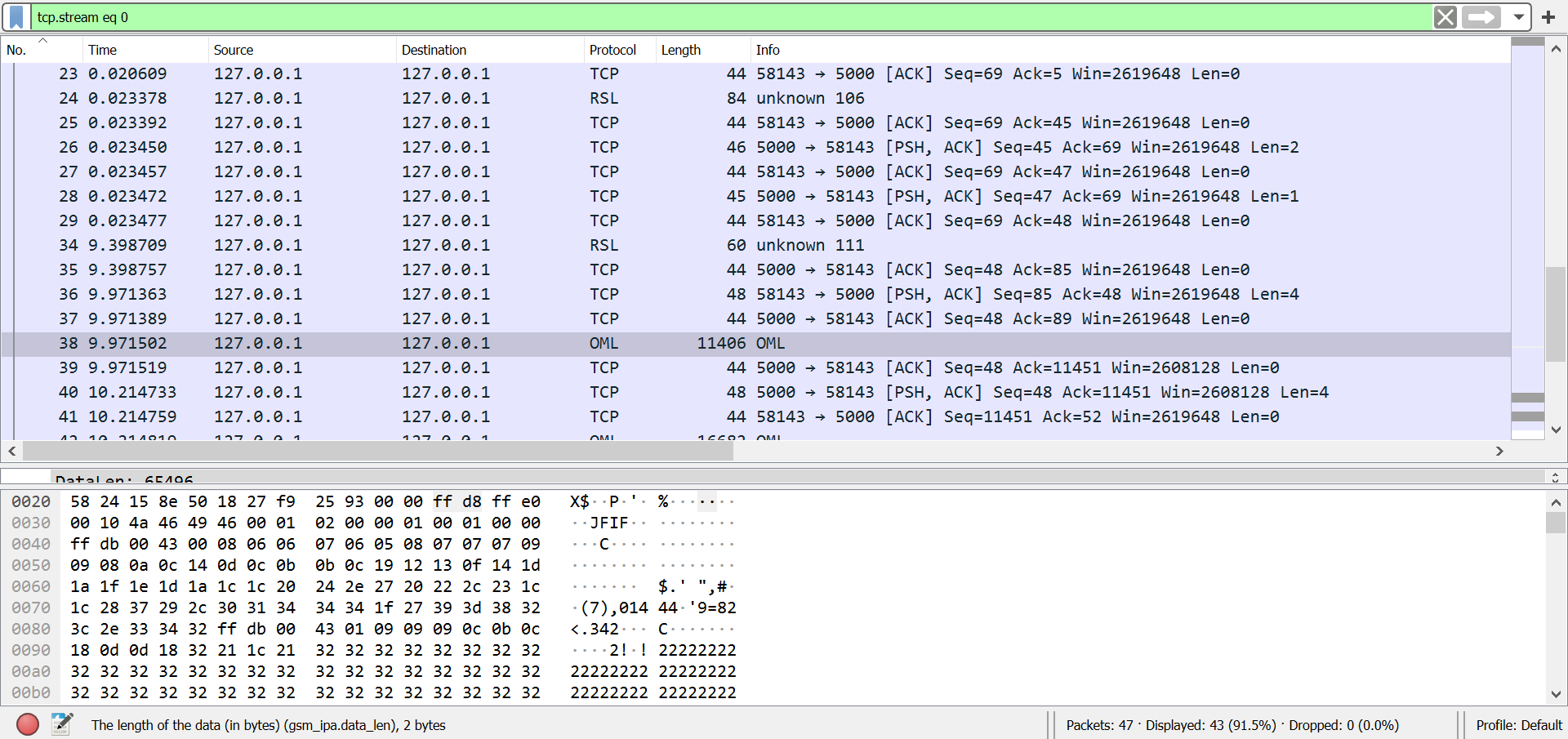
En cliquant sur l’option « Follow TCP Stream » du premier paquet RSL inconnue (« unkown »), on obtient la photo en dessous.



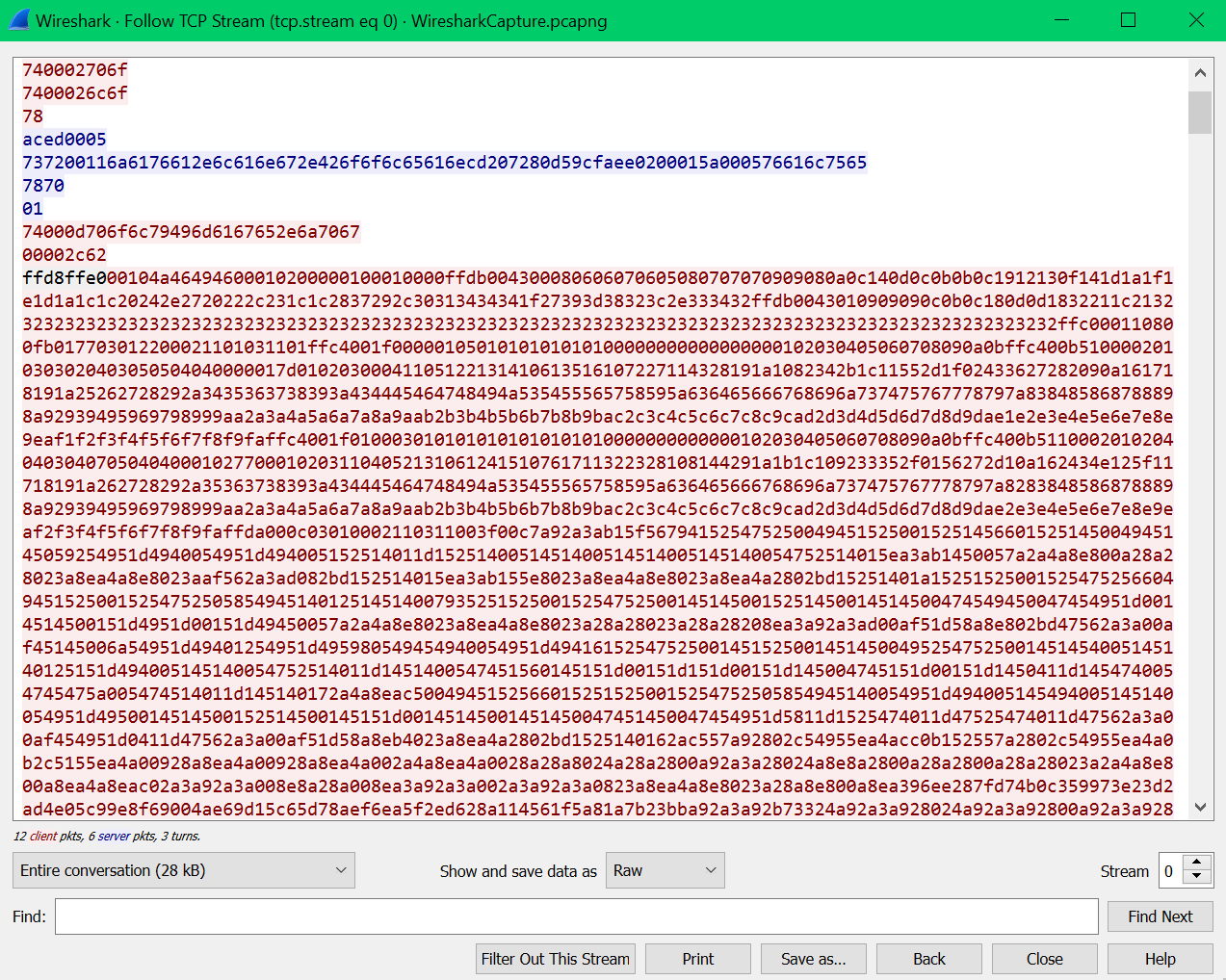
Dans cette photo, on peut observer le nom d’utilisateur de l’utilisateur qui est « roman » et quelques charactères plus tard, on peut observer son mot de passe qui est « 123 ».

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* ”. **(2 points)**

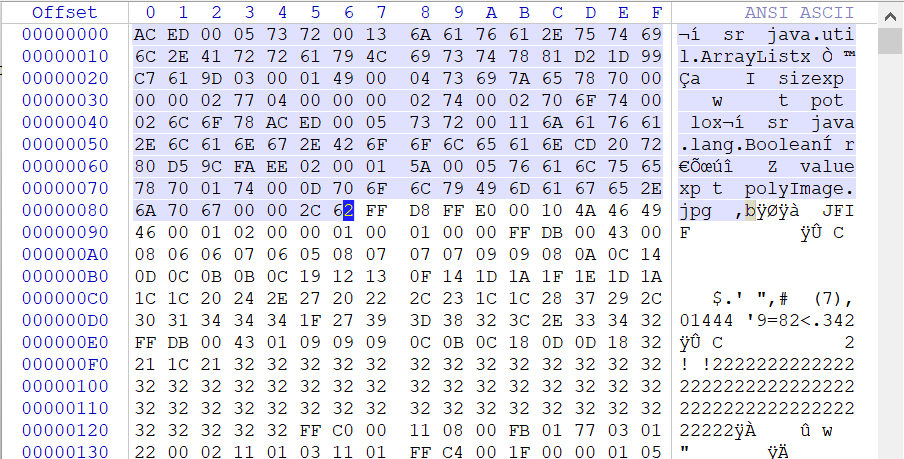
Nous allons regarder l’image envoyée par le client pour cette question vu que le même principe s’applique si nous avions voulu le faire pour l’image traitée. Il faut d’abord sélectionner le paquet sur l’interface de Wireshark.

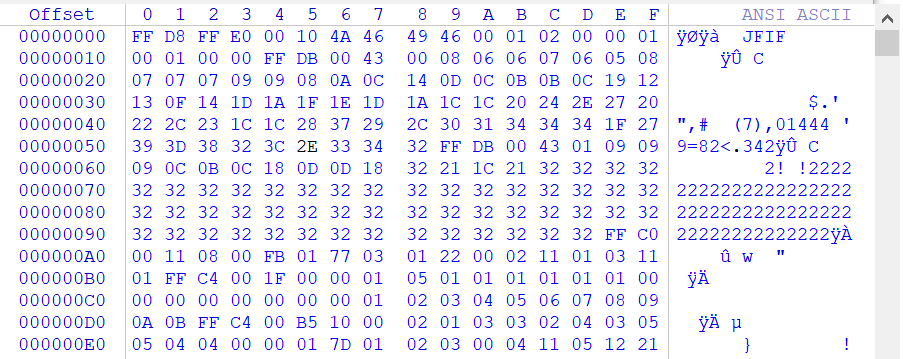


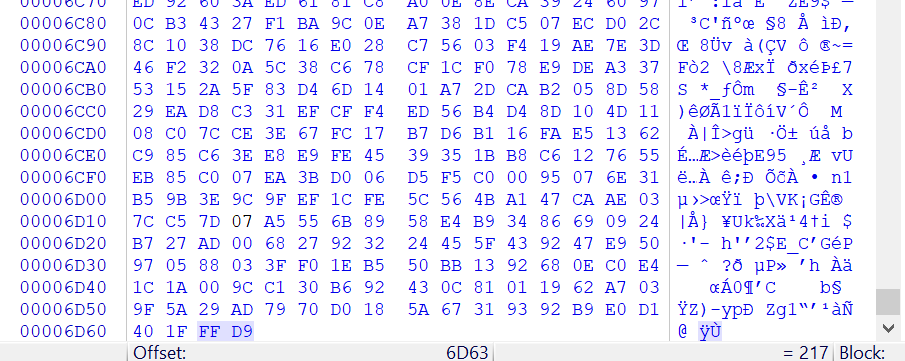
Ensuite, il faut sélectionner l’option « Follow TCP Stream » afin de pouvoir sauvegarder le flot de donnée du paquet.



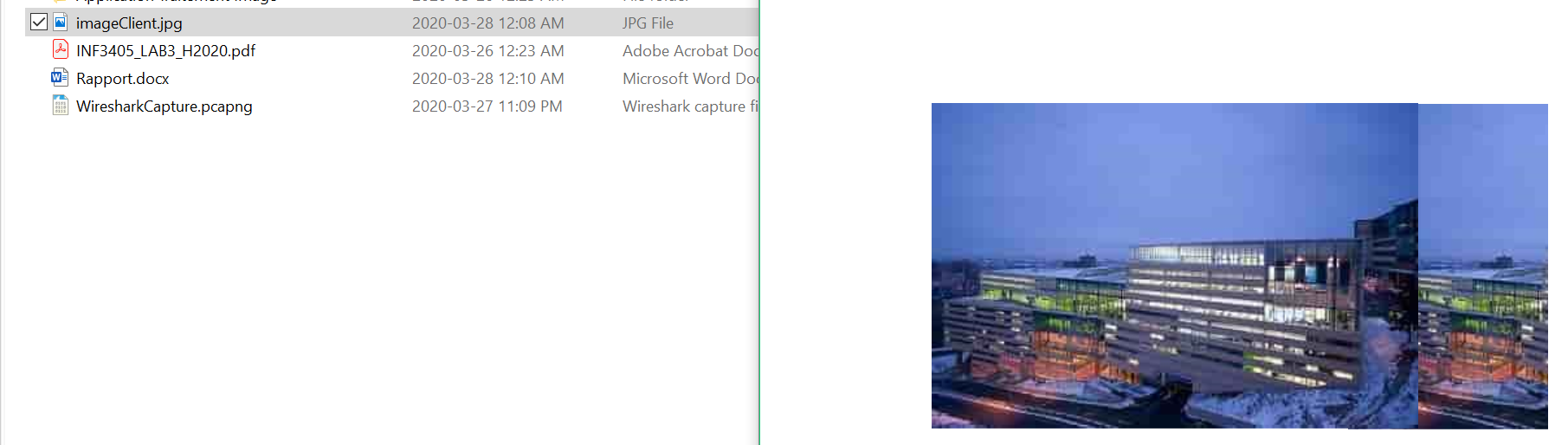
Finalement, à l’aide de WinHex, nous pouvons enlever les informations inutiles pour ne garder que les informations se trouvant entre FF D8 FF E0 et FF D9. Les trois images montrent le segment de données à enlever et le début et la fin de l’image.







Pour montrer que cela à permis d’extraire l’image, nous rajouter l’extension .jpg au fichier afin de pouvoir l’ouvrir comme une image. L’image suivante montre le résultat.



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.2. **(1 point)**

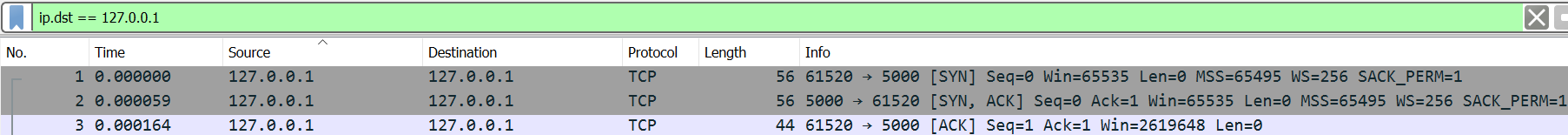
L’application de procure aucune sécurité en ce a trait aux images qui sont envoyées. En effet, on a pu très facilement extraire l’image envoyé de son paquet seulement en sachant le format de l’image et en utilisant des outils faciles à obtenir (Winhex et Wireshark). Il est donc possible pour n’importe qui étant capable d’utiliser ces outils d’extraire le contenu des échanges de notre application de traitement d’image et d’en faire ce qu’il souhaite. Une façon de remédier à cette faille de sécurité, serait de permettre à l’application du traitement d’image d’encrypter le fichier contenant l’image avant de l’envoyer et de le décrypter à la réception. De cette façon, les données ne sont exposées que dans l’application et non dans la couche de transport qui est facilement accessible.

9. Analyse d’une application client-serveur “secrète”

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 se nomme cet échange? Dans le cas d'UDP, est-ce que ce même échange à lieu? Pourquoi? **(0.5 point)**

**Mode 1**

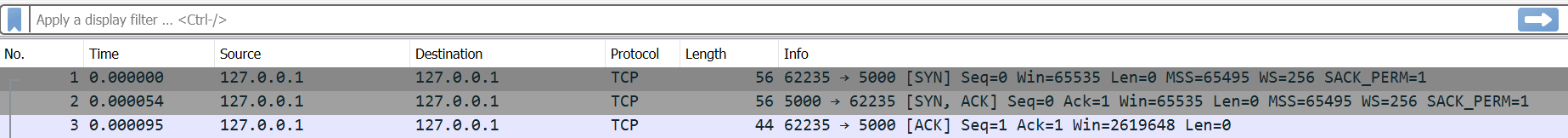
TCP est le protocole utilisé et voici le premier échange entre le client et le serveur lors de l’initialisation de la connexion.



Cet échange s’appel le *Three-way handshake* [1]. Il sert à synchroniser les numéros de séquence entre le client et le serveur afin d’établir la connexion entre ces derniers.

**Mode 2**

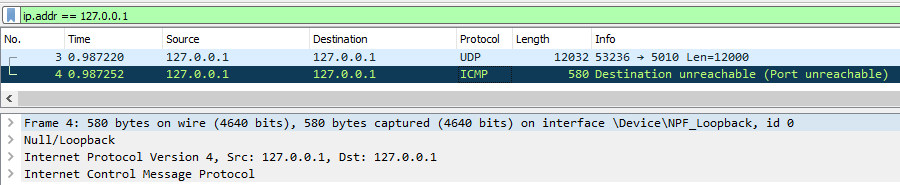
TCP est le protocole utilisé et voici le premier échange entre le client et le serveur lors de l’initialisation de la connexion.



Cet échange s’appel le *Three-way handshake* [1]. Il sert à synchroniser les numéros de séquence entre le client et le serveur afin d’établir la connexion entre ces derniers.

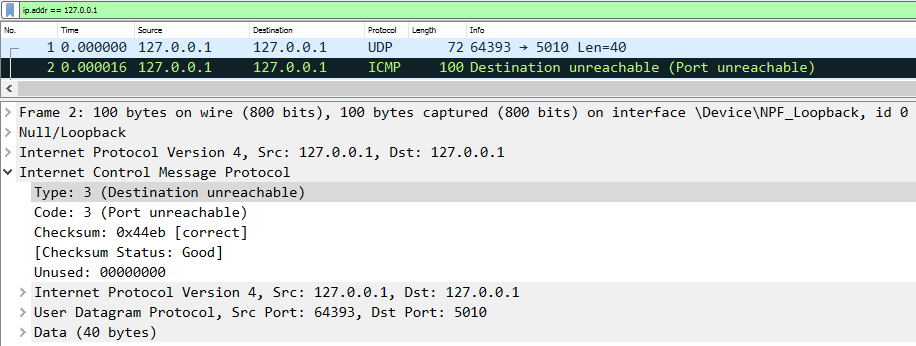
**Mode 3**

UDP est le protocole utilisé et l’échange n’a pas eu lieu car, contrairement au protocole TCP, le protocole UDP ne vérifie pas que la connexion soit établie avec la destination; il ne fait qu’envoyer son message. Cependant, on reçoit un paquet ICMP qui nous indique qu’il a eu un problème de réseau empêchent la livraison du paquet (port inaccessible).



**Mode 4**

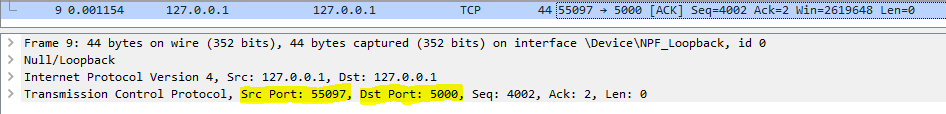
UDP est le protocole utilisé et l’échange n’a pas eu lieu car, contrairement au protocole TCP, le protocole UDP ne vérifie pas que la connexion soit établie avec la destination; il ne fait qu’envoyer son message. Cependant, on reçoit un paquet ICMP qui nous indique qu’il a eu un problème de réseau empêchent la livraison du paquet (port inaccessible).



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

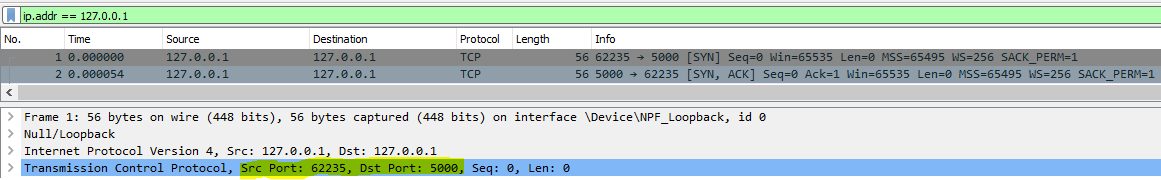
**Mode 1**

Le port source est 55097 et le port de destination est 5000.



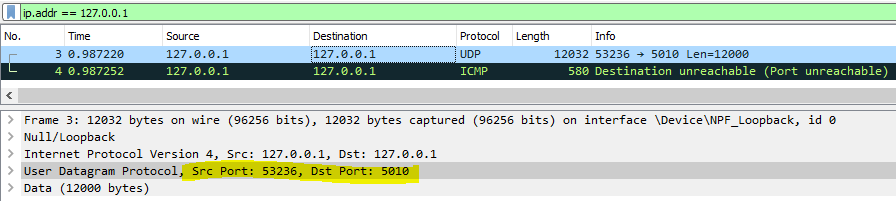
**Mode 2**

Le port source est 62235 et le port de destination est 5000.



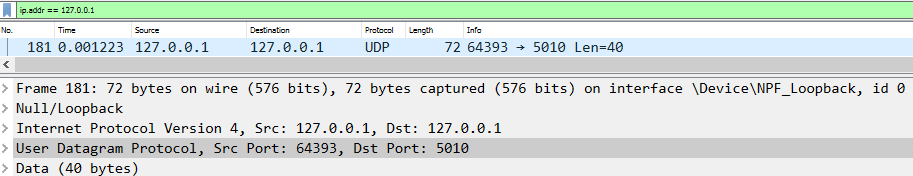
**Mode 3**

Le port source est 53236 et le port de destination est 5010.



**Mode 4**

Le port source est le 64393 et le port destination est le port 5010.

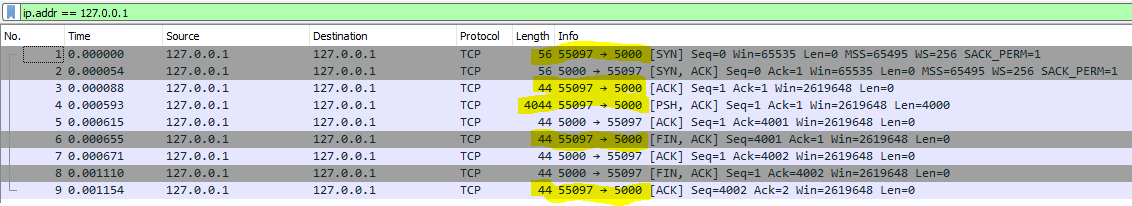




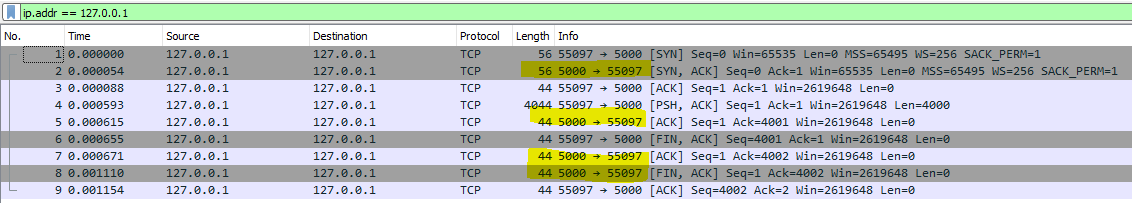
3) Combien de paquets et d’octets contenant des données ont été envoyés par le client vers le serveur? Par le serveur vers le client? Montrer où vous avez trouvé cette information. **(0.5 point)**

**Mode 1**

Le client a envoyé 5 paquets TCP avec une taille totale de 4232 octets, ce qui fait que le client a envoyé 4232 – 5\*20 = 4132 octets.

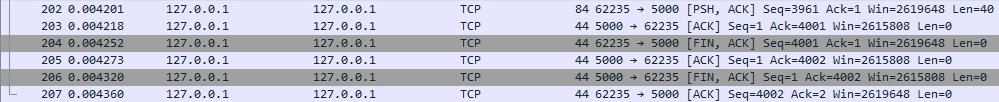
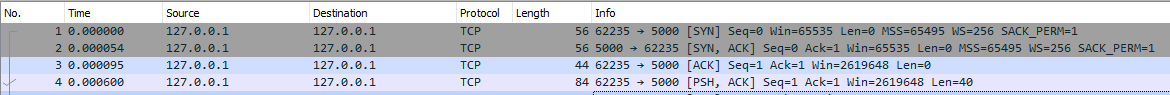


Le serveur a envoyé 4 paquets TCP avec une taille totale de 188 octets, ce qui fait que le serveur a envoyé 188 – 4\*20 = 108 octets.

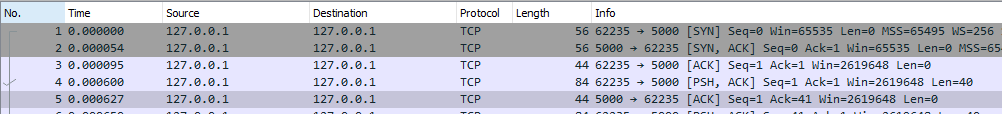


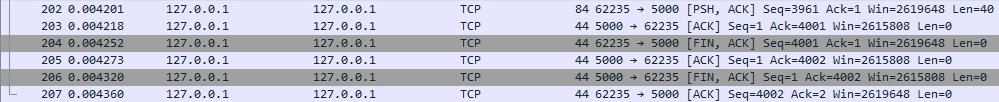
**Mode 2**

Le client a envoyé 104 paquets TCP avec une taille totale de (56+44+44+44+100\*84) 8588 octets, ce qui fait que le client a envoyé 8588 – 104\*20 = 6508 octets.



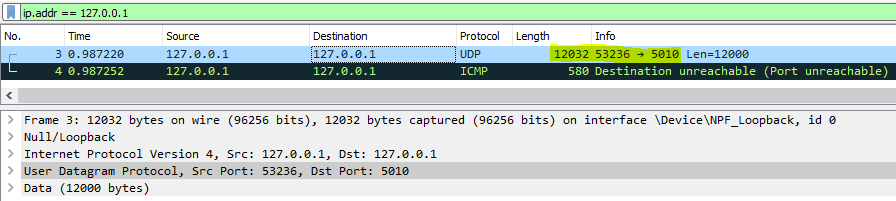
Le serveur a envoyé 103 paquets TCP avec une taille totale de (56+102\*44) 4544 octets, ce qui fait que le serveur a envoyé 4544 – 103\*20 = 2484 octets.





**Mode 3**

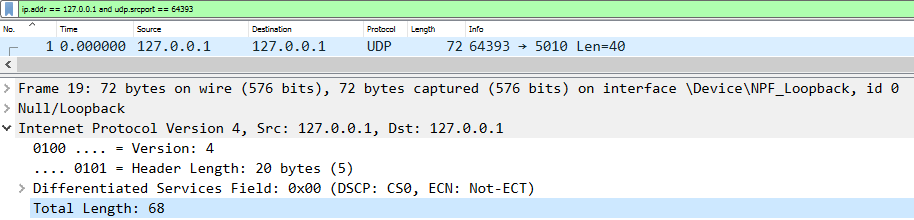
Le client a envoyé un paquet UDP avec une taille totale de 12032 octets, ce qui fait que le client a envoyé 12032 - 20 = 12012 octets



Le serveur n'a envoyé aucun paquet au client.

**Mode 4**

Le client a envoyé 300 paquets UDP de taille (length) 72 et ayant un en-tête de taille 20. Il y a alors 15600 octets de données qui a été envoyés. Les paquets IMCP ne sont pas comptés car ils ne peuvent pas être considérés comme provenant du client pour le serveur ou provenant du serveur pour le client.



Le serveur n’a envoyé aucun paquet au client.

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**

Selon nous, le client envoie une image ou un fichier volumineux au serveur, car un des paquets qu’il envoie au serveur contient 4024 octets de données, ce qui est plus grand qu’un paquet « habituel ». On pense que le client fait 3 ou 4 itérations pour envoyer ces données.

**Mode 2**

Le client envoie plusieurs petits paquets au serveur et le serveur affirme au client la réception de paquet. Le client fait environ 100 itérations pour envoyer ces données.

**Mode 3**

Le client envoie un fichier volumineux au serveur et il le fait avec une seule itération.

**Mode 4**

Le client tente d’abord d’envoyer des informations au serveur sans succès pendant 100 itérations. Puis, il réussit à envoyer, pendant 200 itérations, des informations au serveur.

**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)**

Performance = vitesse de réception de toutes les données?

Le mode 1 envoie la totalité de ses données en 0.001154 second, tandis que le mode 2 envoie la totalité de ses données en 0.00436 second, ce qui est 3.8 fois plus lent que le mode 1. La différence entre les 2 modes est que le mode 1 utilise un paquet pour envoyer son fichier volumineux, tandis que le mode 2 utilise plusieurs paquets pour envoyer ce même ficher. Alors, le mode 1 est plus performant, car il transmet la totalité de l’information plus rapidement 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)**

Le mode 3 envoie la totalité de ses données en 0.000034 second, tandis que le mode 4 envoie la totalité de ses données en 0.008612 second, ce qui est 253 fois plus lent que le mode 3. La différence entre les 2 modes est que le mode 3 utilise un seul paquet pour envoyer la totalité de ses données, tandis que le mode 4 utilise plusieurs paquets pour envoyer ses données. Le mode 3 est plus performant que le mode 4, car le mode 3 transmet la totalité de l’information plus rapidement que le mode 4.

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

Les modes 1 et 2 sont les plus fiables, car ces modes utilisent le protocole TCP, alors si un paquet n’est reçu, ce protocole va renvoyer ce paquet pour assurer que chaque paquet soit livré à son destinataire.

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)**

Cet échange indique la terminaison de la connexion entre le client et le serveur. Alors, le client commence par envoyé un paquet FIN, ACK au serveur (ce paquet indique que le client veut couper la connexion entre les deux) et le server confirme au client la réception de ce paquet. Finalement, le serveur envoie à son tour un paquet FIN, ACK au client et le client confirme la réception de ce paquet avec un paquet ACK.