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)**

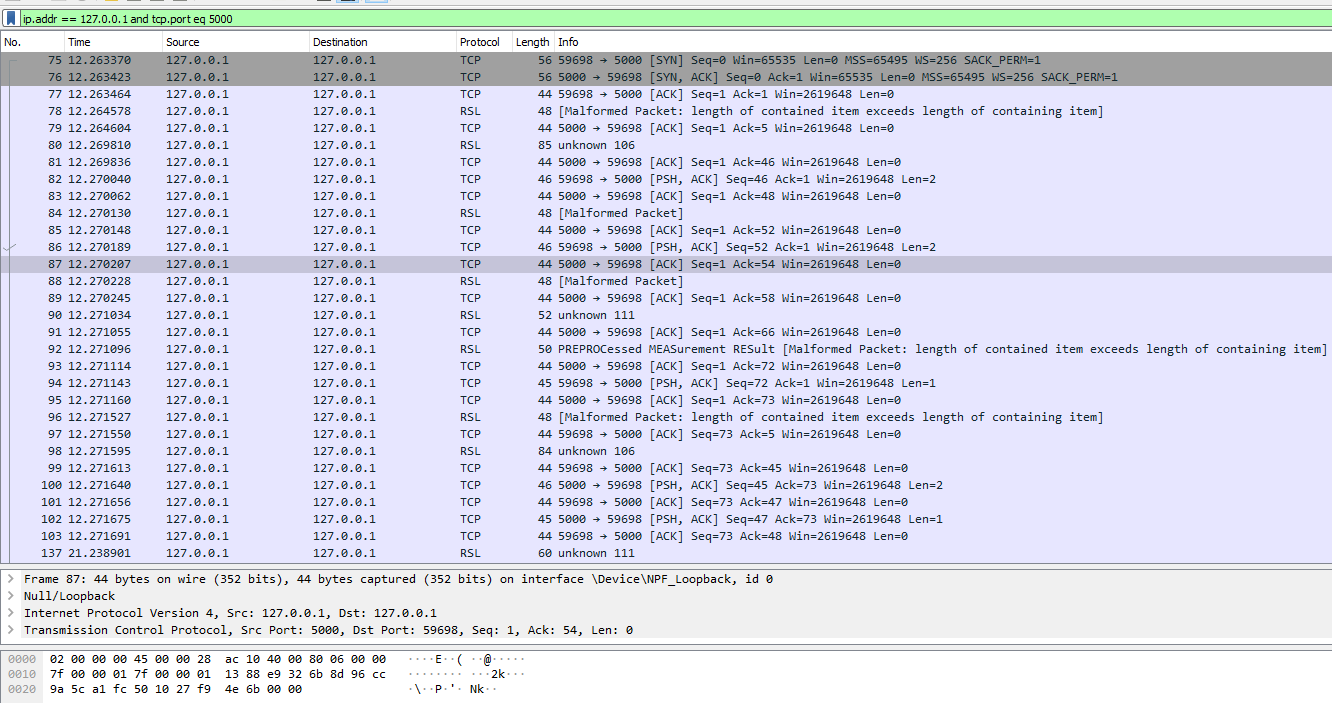
(pas sûr) Je mettrai pas le tcp.port parce que je pourrais en choisir un autre mais le 127.0.0.1 reste vrai parce qu’il est mentionné dans l’énoncé.

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.

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)**

**Pas sûr non plus, car ca peut êre RSL (radio signalling link), mais je pense pas. Par contre le protocol OML (transmission control protocol) contient tous les données à mon avis, car il contient beaucoup de données. Non pour RSL car il vient de la couche 3 (wiki). Je dirais pas OML cuz il s’occupe juste du transfert dimage i guess. C’est vraiment TCP qui permet la communication.**

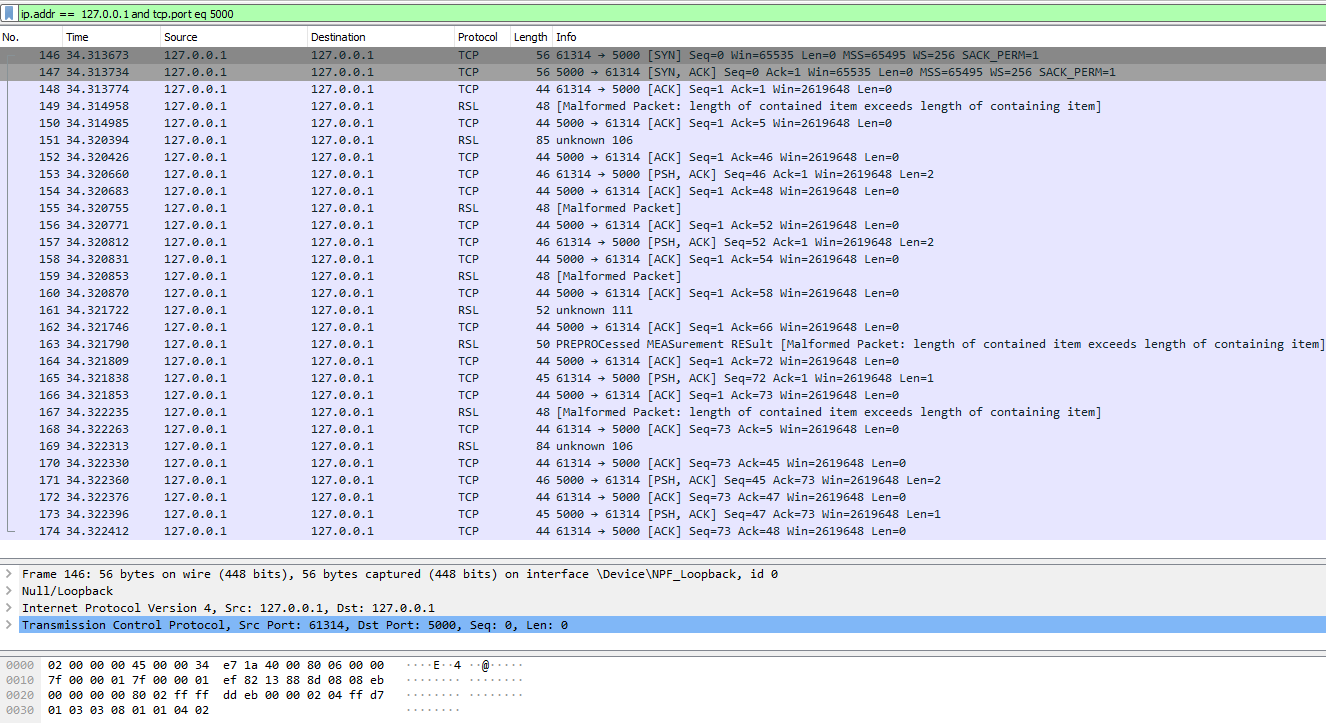
TCP



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)**

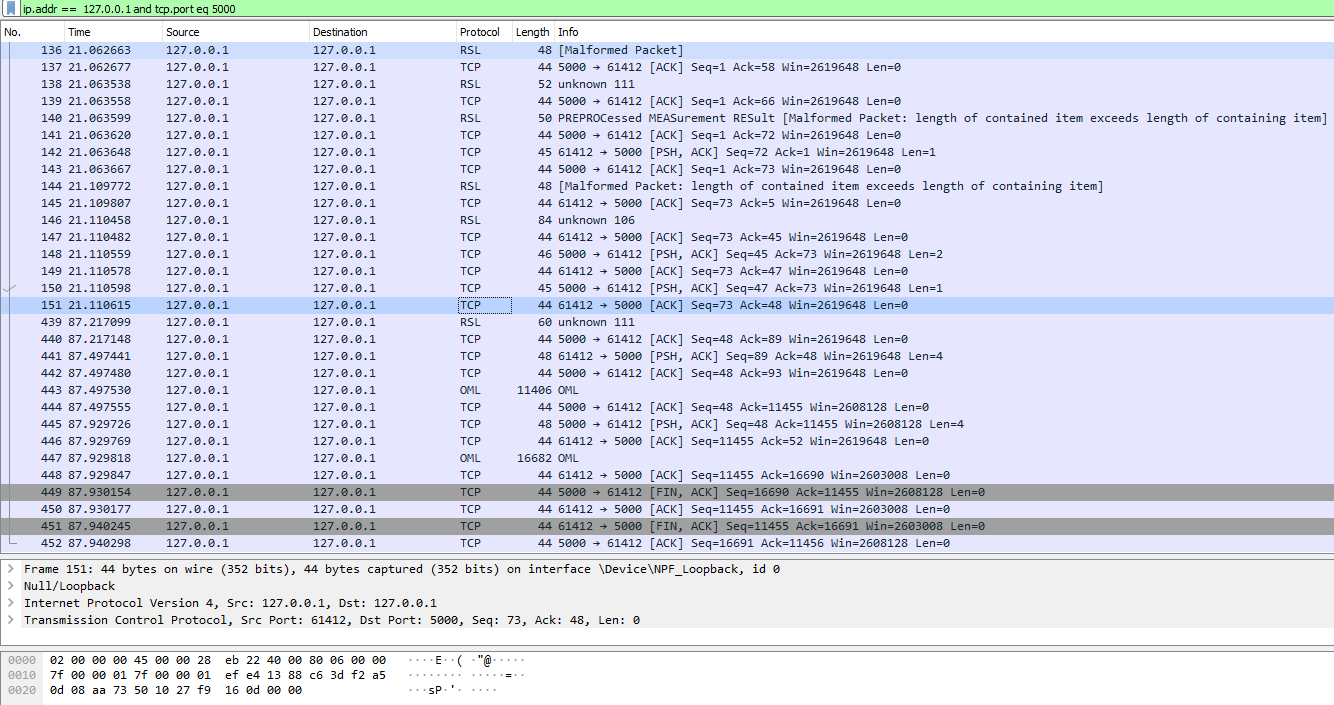
**Pas sûr**

Image lors de connexion du client au serveur :



Il y a 21 paquets TCP et 8 RSL (5 malformé et 3 inconnues)

Image après la réception de l’image sobelisé



Il y a 11 nouveaux paquets TCP, ce qui donne un total de 32 paquets TCP.

Les 21 premiers paquets TCP établies la connexion entre le client et le serveur. Parmi ceux-là, le client envoie au serveur 9 paquets TCP et 6 RSL et le serveur envoie 12 paquets TCP et 2 RSL au client.

Ensuite, le client envoie un paquet RSL, 5 paquets TCP et un paquet OML (image originale).

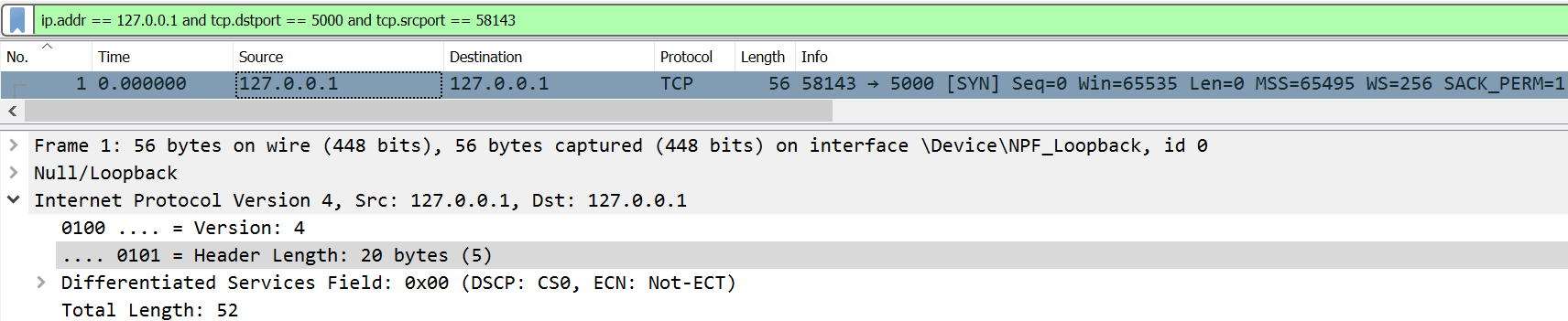
Le serveur envoie 6 paquets TCP et un paquet OML (image sobelisé).

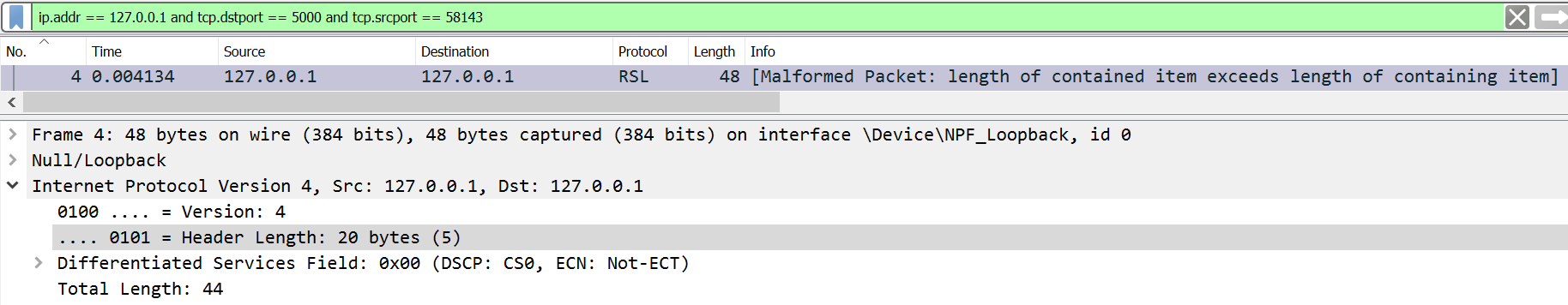
Alors, le client a envoyé 14 paquets TCP, 7 paquets RSL et 1 paquet OML, ce qui donne un total de 9 (données TCP) + 83 (données RSL incluant les malformés et les inconnues) + 11362 (données OML) = 11454 données.

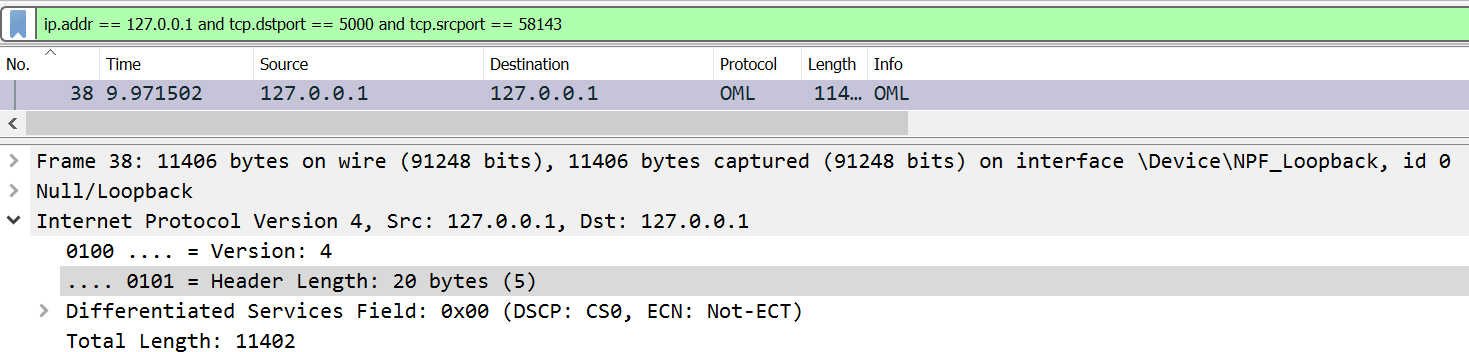
Le serveur a envoyé 18 paquets TCP, 2 paquets RSL et 1 paquet OML, ce qui donne un total de 7 (données TCP) + 44 (données RSL incluant les malformés et les inconnues) + 16638 (données OML) = 16689 données.

CE QUE HAKIM VOIT

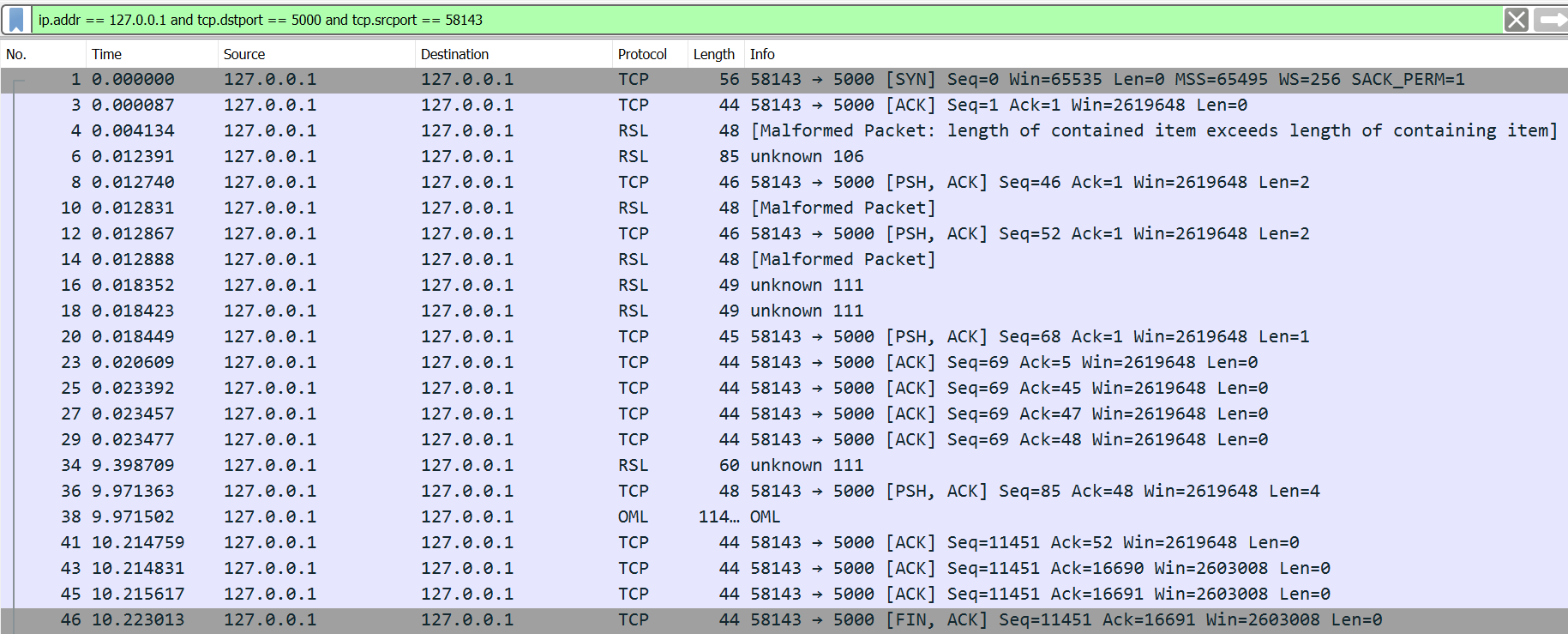
Dans l’ensemble, nous avons vu qu’il n’y a que des paquets TCP, RSL et OML qui sont échangés entre le client et le serveur. En utilisant Wireshark, nous pouvons déterminer la taille des en-têtes de chacun des paquets et ainsi calculer la quantité de données dans chacun des paquets en soustrayant la taille 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).





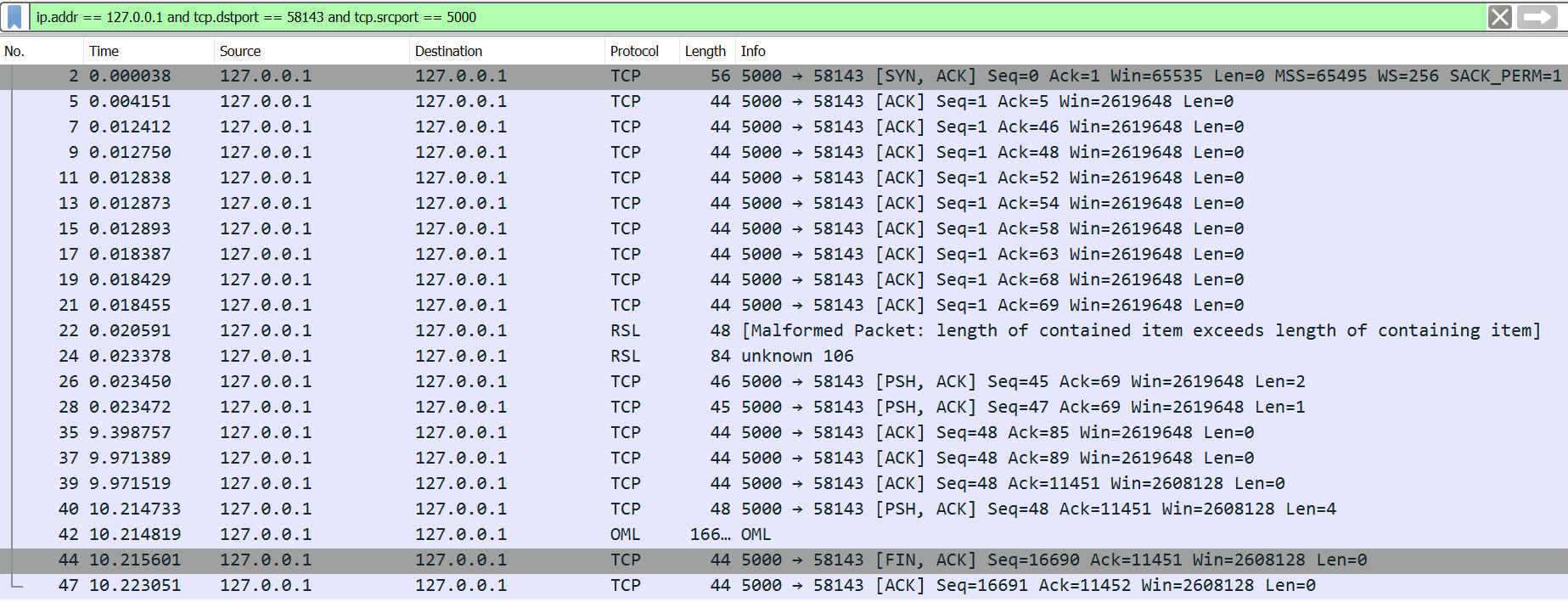


L’image suivante montre les paquets que le client envoit au serveur (nous avons utilisé le filtre « ip.addr == 127.0.0.1 and tcp.dstport == 5000 and tcp.srcport == 58143 » afin d’identifier les paquets en direction du serveur).



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.

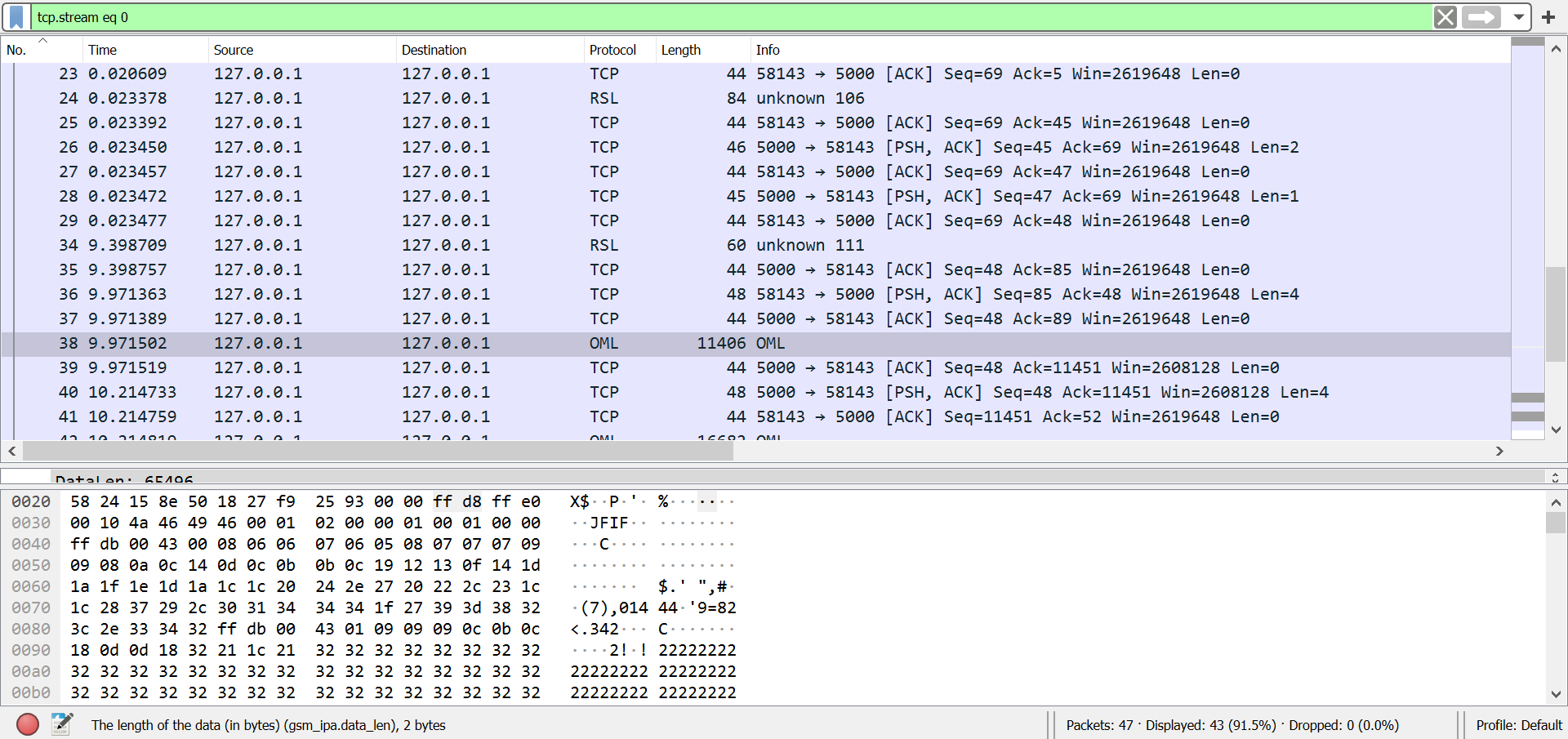
Hakim : (Voir <https://osqa-ask.wireshark.org/questions/24699/tcp-packet-length-was-much-greater-than-mtu>)

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)**

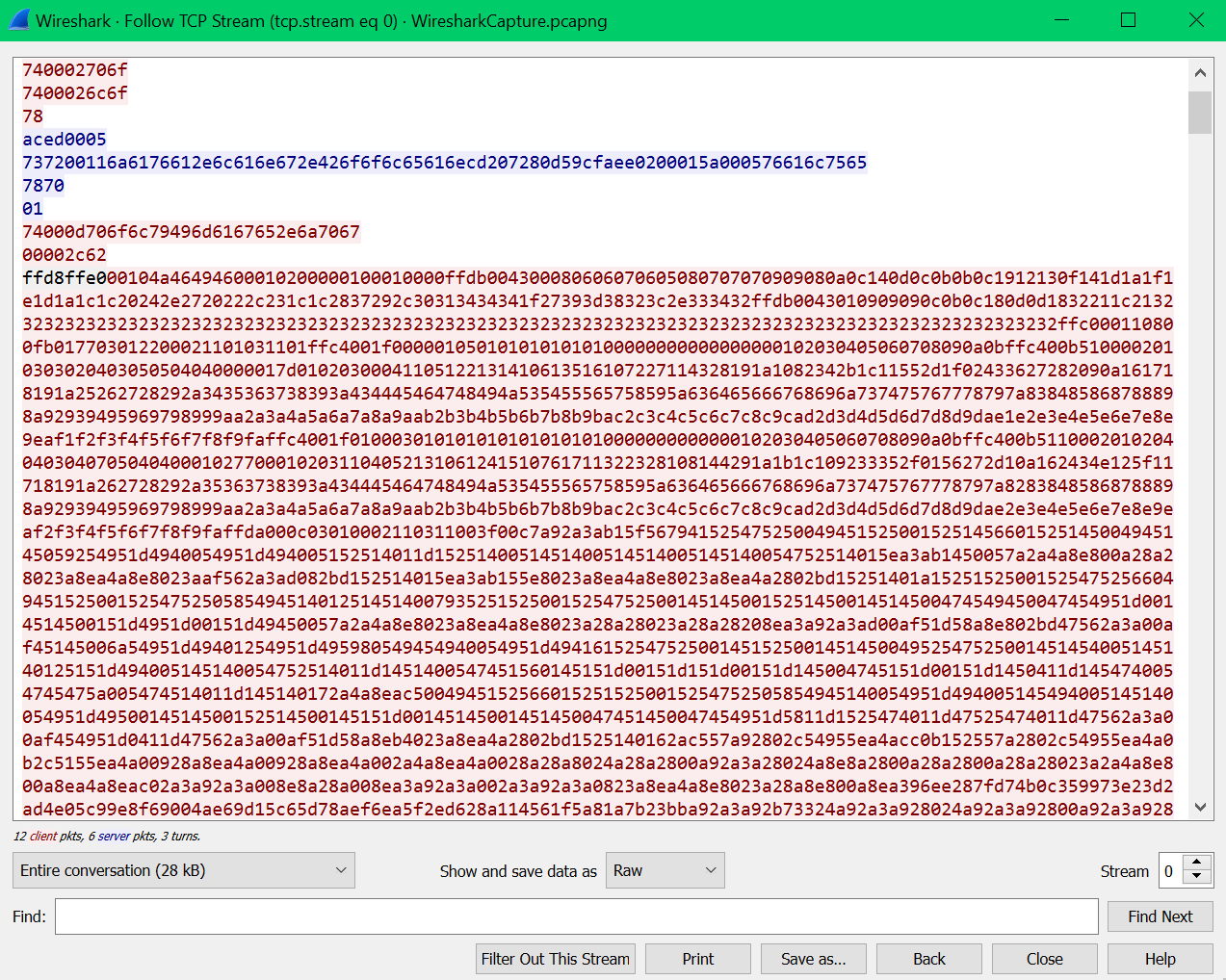
What the fuck

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)**

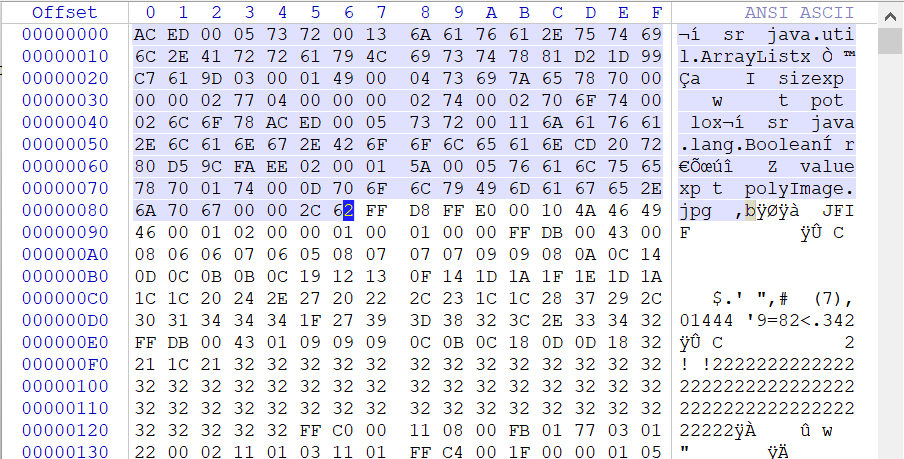
Nous allons regarder l’image envoyée par le client pour cette question vu que le même principe s’applique si noud avions voulu le faire pour l’image traitée. Il faut d’abord sélectionner le paquet sur l’interface de Wireshark.

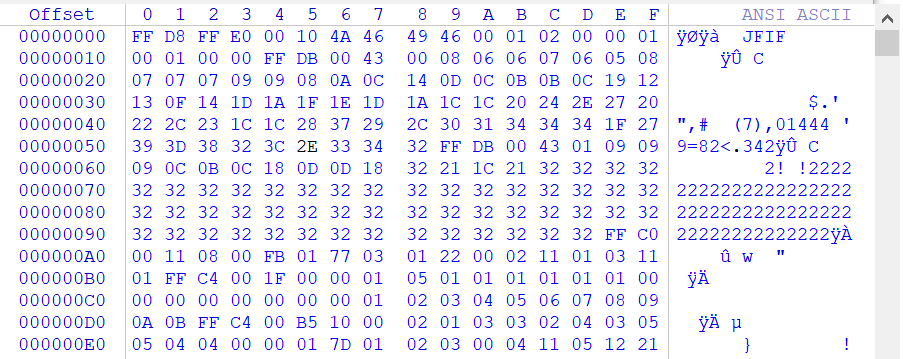


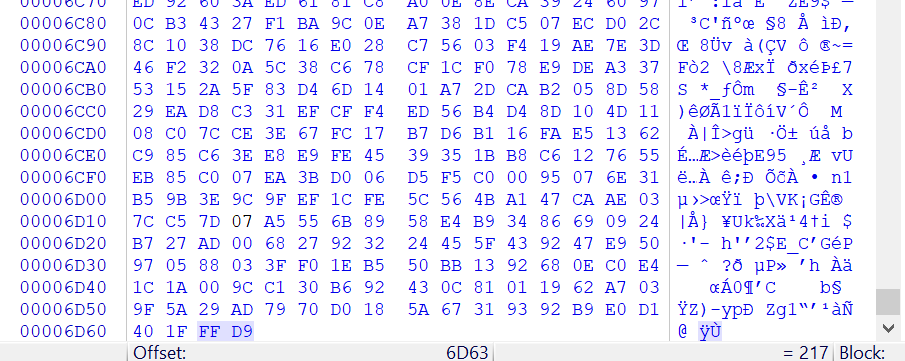
Ensuite, il faut sélectionner l’option « Follow TCP Stream » afin de pouvoir sauvegarder le flot de donnée du paquet.



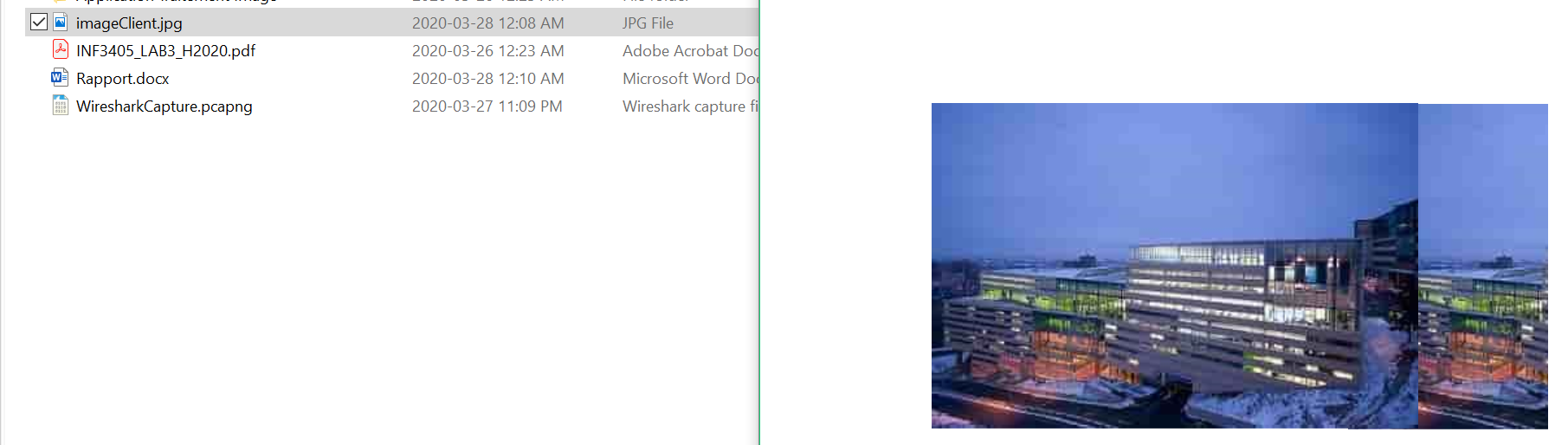
Finalement, à l’aide de WinHex, nous pouvons enlever les informations inutiles pour ne garder que les informations se trouvant entre FF D8 FF E0 et FF D9. Les trois images montrent le segment de données à enlever et le début et la fin de l’image.







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.

**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? **(0.5 point)**

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

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)**

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)**

**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)**

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)**

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

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)**