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 du port 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. RSL et OML font partie de TCP et ne sont donc pas considérés comme des protocoles à part.

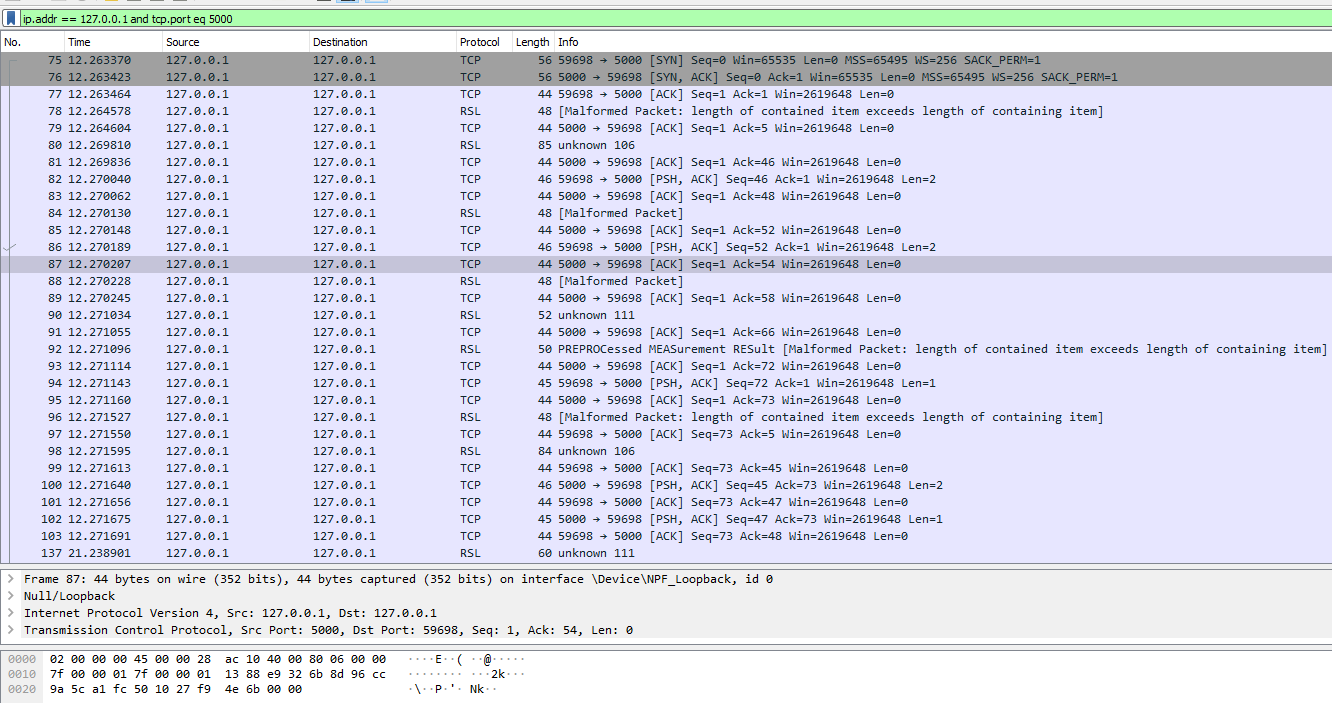


Figure 1: Vue d'ensemble de la communication entre le client et le serveur

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)**

À l’aide des filtres et de l’outil de statistiques de Wireshark, nous pouvons facilement visualiser le nombre de paquets et d’octets de données envoyées par le client et le serveur. Pour trouver le nombre d’octets de données envoyés, il suffit de regarder le numéro de séquence du dernier paquet envoyé par le client ou le serveur. En effet, puisque le protocole employé est le TCP, nous pouvons nous fier aux numéros de séquence pour trouver le nombre d’octets de données, car ce dernier est incrémenté pour chaque octet de donnée qui transige.

Pour le serveur, il faut d’abord appliquer le filtre « ip.addr == 127.0.0.1 and tcp.dstport == 58143 and tcp.srcport == 5000 » afin d’identifier les paquets en provenance du serveur et en direction du client. Voici l’affichage de Wireshark pour, respectivement, l’application du filtre et les statistiques :

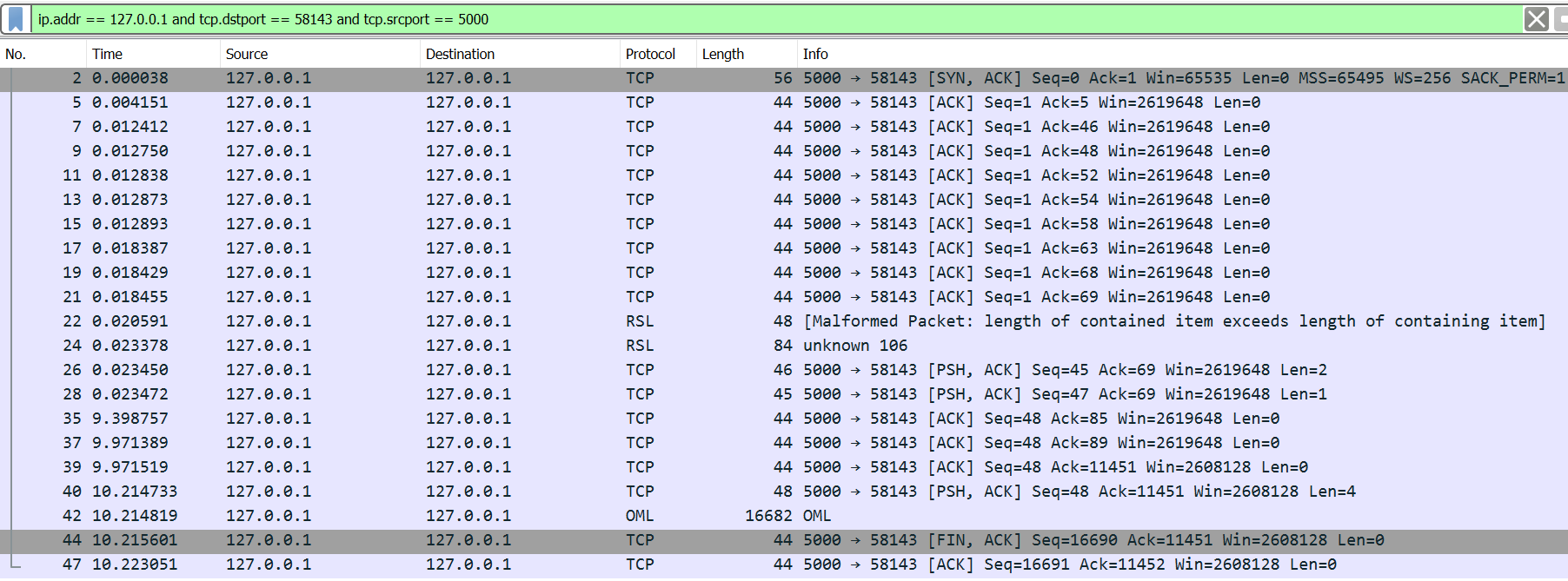




Figure 2: Application du filtre « ip.addr == 127.0.0.1 and tcp.dstport == 58143 and tcp.srcport == 5000 »

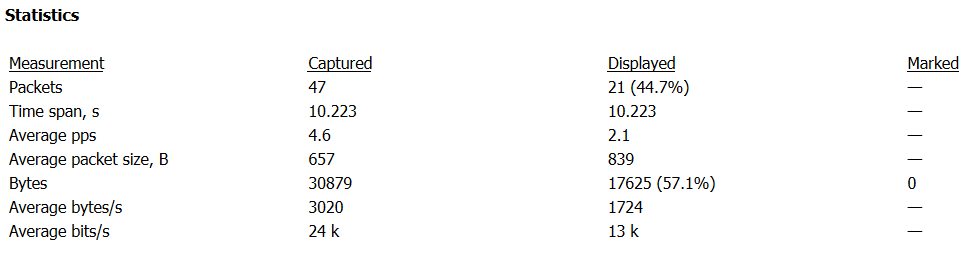




Figure 3: Affichage des statistiques de Wireshark pour les paquets envoyés depuis le serveur vers le client

On peut donc voir que le serveur a envoyé 21 paquets et que le dernier numéro de séquence est 16691. Cependant, ce numéro de séquence est aussi incrémenté lors des échanges servant à initialiser la connexion dans le cas du serveur puisque c’est lui qui doit répondre à la demande de connexion du client. Il faut alors soustraire 1 à ce nombre car, lors de l’initialisation, aucune donnée « utile » n’est échangée entre le client et le serveur. Le serveur a alors envoyé 16690 octets de données au client.

Pour le client, il faut d’abord appliquer le filtre « ip.addr == 127.0.0.1 and tcp.dstport == 5000 and tcp.srcport == 58143 » afin d’identifier les paquets en provenance du client et en direction du serveur. Voici l’affichage de Wireshark pour, respectivement, l’application du filtre et les statistiques :

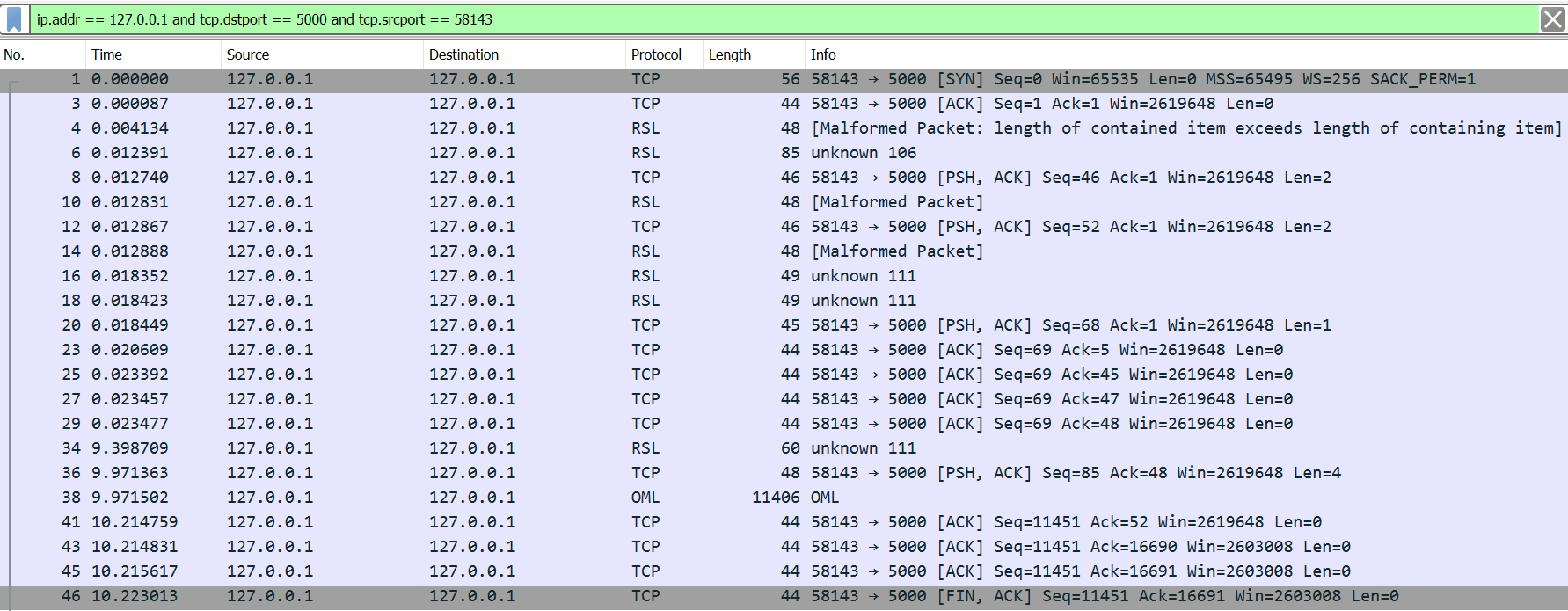




Figure 4: Application du filtre le filtre « ip.addr == 127.0.0.1 and tcp.dstport == 5000 and tcp.srcport == 58143 »

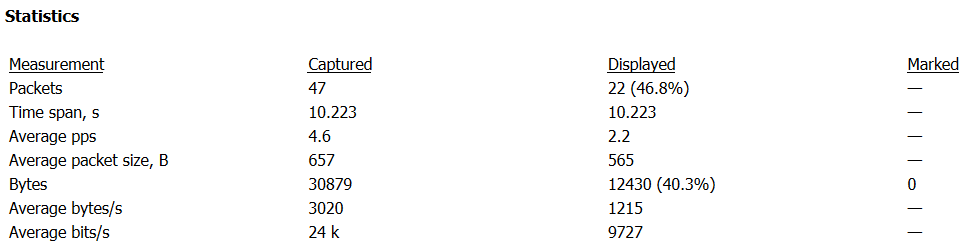




Figure 5: Affichage des statistiques de Wireshark pour les paquets envoyés depuis le client vers le serveur

On peut donc voir que le client a envoyé 22 paquets et que le dernier numéro de séquence est 11451. Cependant, ce numéro de séquence est aussi incrémenté lors des échanges servant à initialiser la connexion et mettre fin à la connexion dans le cas du client puisque c’est lui qui a amorcé l’échange. Il faut alors soustraire 2 à ce nombre car, lors de l’initialisation et la fin de la connexion, aucune donnée « utile » qui est échangée entre le client et le serveur. Le client a alors envoyé 11449 octets de données au serveur.

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 (*length*) de 11406 octets et de 16682 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 à partir du paquet utilisé pour authentifier l’utilisateur.

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 figure ci-dessous.

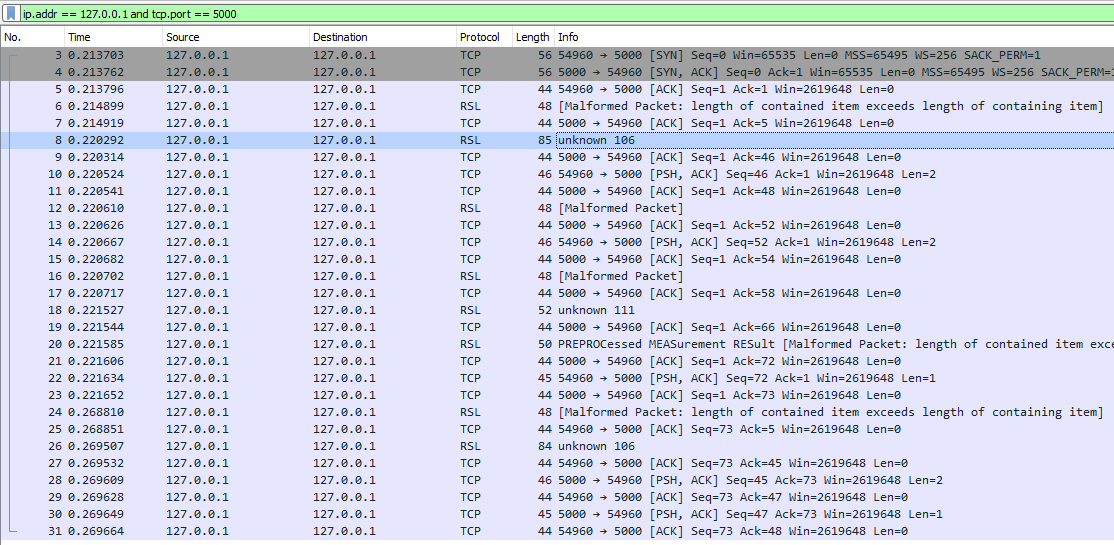


Figure 6: Première série de paquets lors de la connexion du client

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

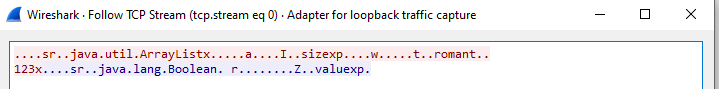


Figure 7: Nom d'utilisateur et mot de passe dans un paquet

Dans cette figure, 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.

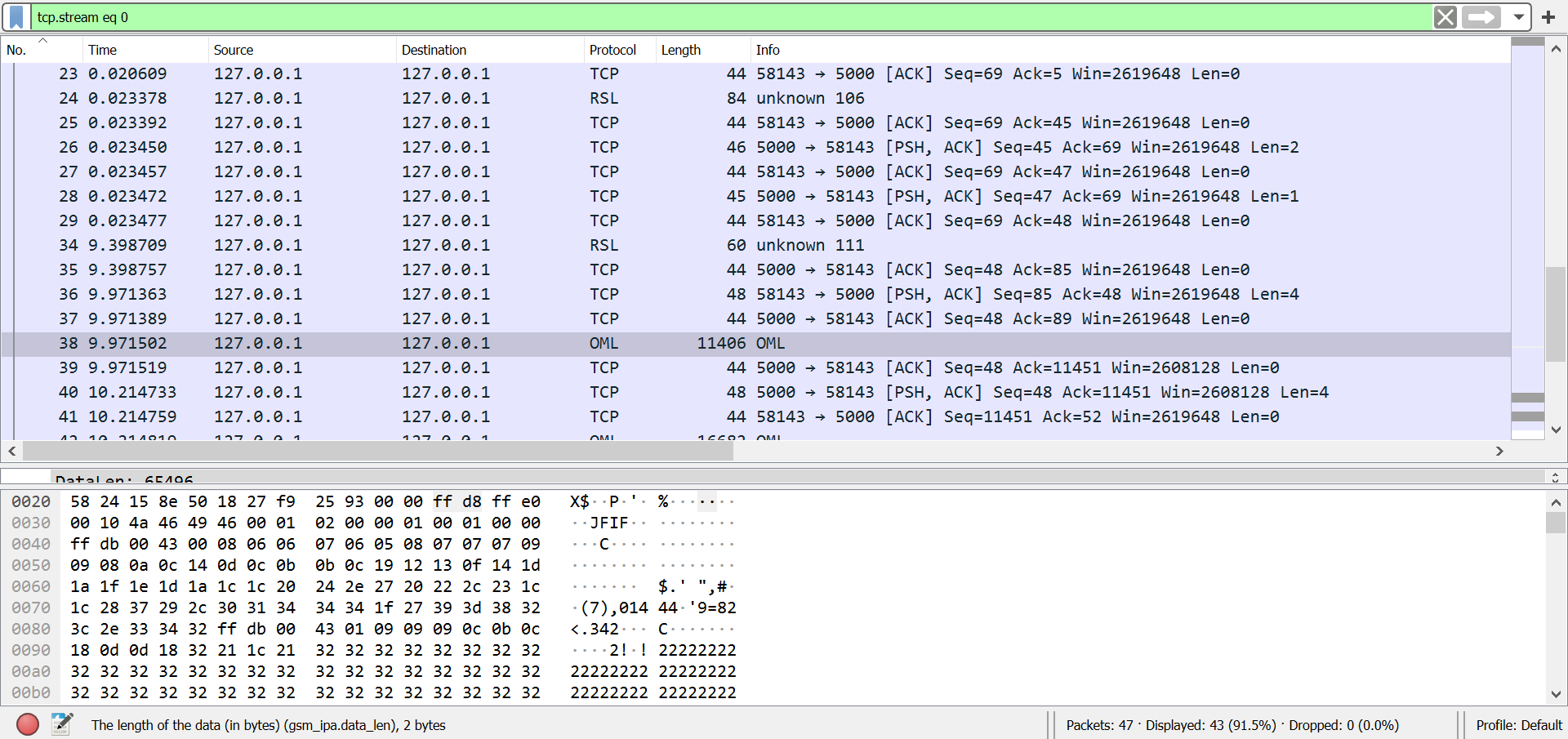


Figure 8: Paquet contenant l'image

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

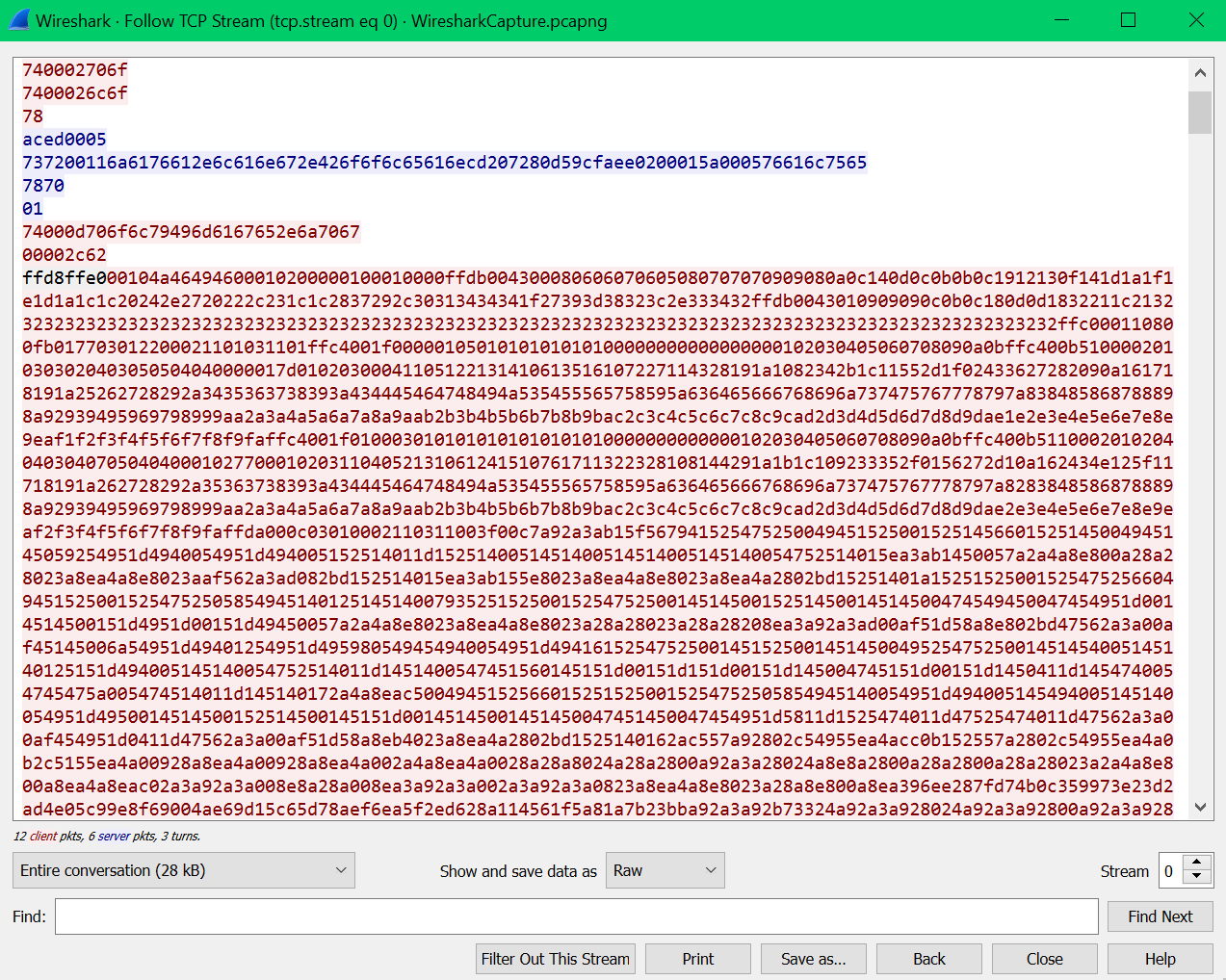




Figure 9: Contenu du paquet contenant l'image

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 suivantes montrent le segment de données à enlever et le début et la fin de l’image.

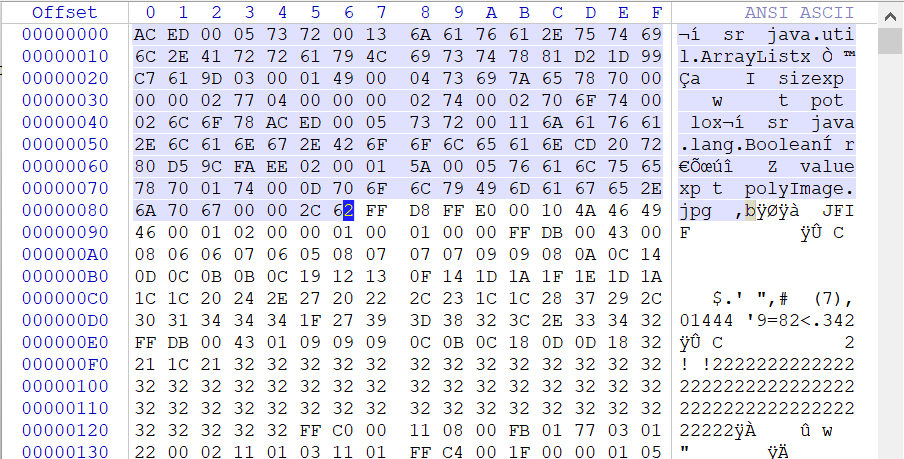




Figure 10: Données représentant l'image et les données non pertinentes à l'image

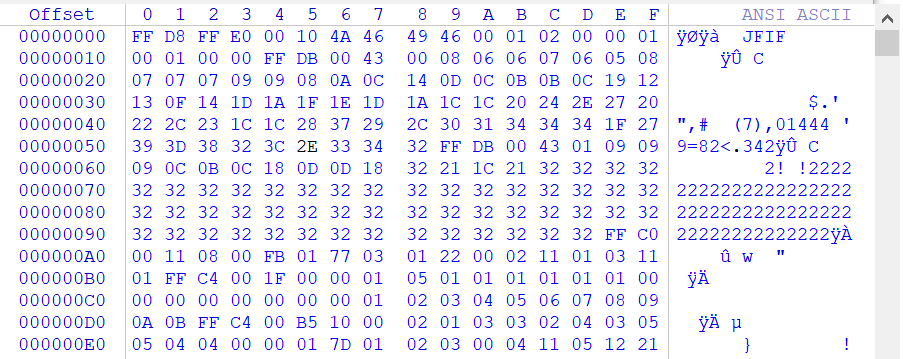




Figure 11: Données représentant l'image sans les données non pertinentes à l’image

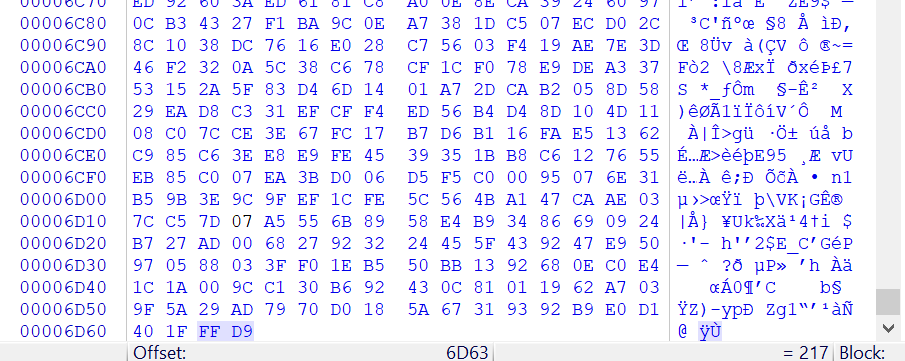




Figure 12: Fin des données représentant l'image

Pour montrer que cela a permis d’extraire l’image, nous avons rajouté l’extension .jpg au fichier afin de pouvoir l’ouvrir comme une image. La figure suivante montre le résultat.

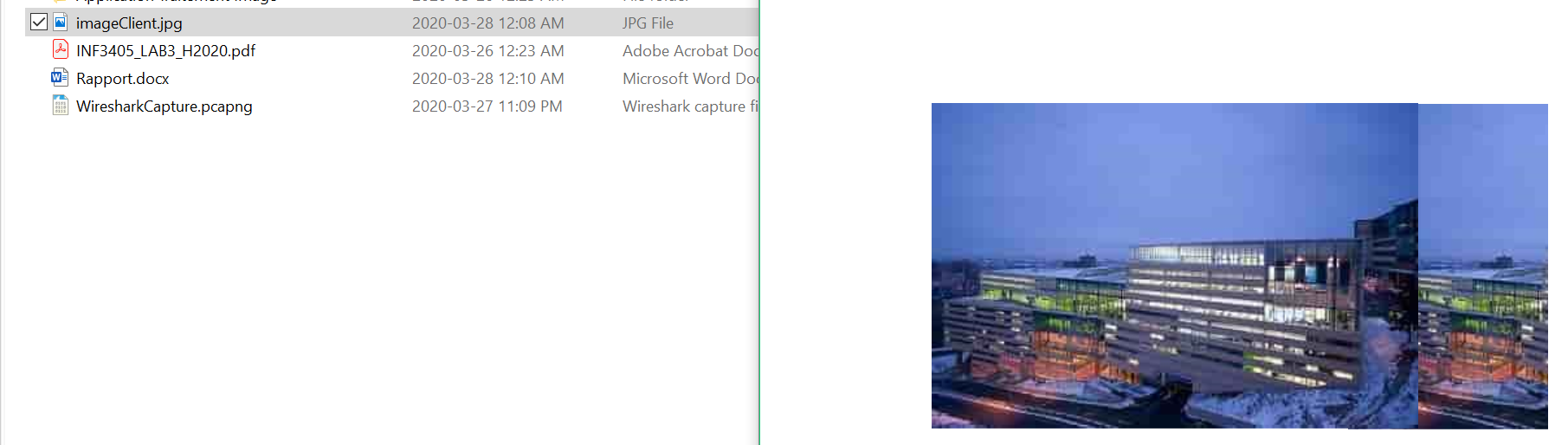


Figure 13: Résultat de l'ouverture en .jpg de l'image extraite d'un paquet

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 et aux données 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 en va de même pour le nom d’utilisateur et le mot de passe. 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 contenu de ses échanges, c’est-à-dire, les octets de données 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.

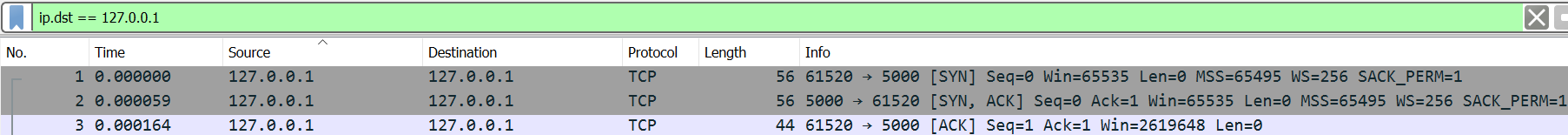


Figure 14: Three-way handshake du protocole TCP (Mode 1)

Cet échange est appelé le *Three-way handshake*. Il sert à synchroniser les numéros de séquence entre le client et le serveur afin d’établir la connexion entre ces derniers. Il débute par le client qui envoie un paquet SYN afin d’effectuer une demande de connexion avec le serveur. Ensuite, le serveur répond avec un paquet SYN-ACK signifiant qu’il accepte que le client se connecte. Finalement, le client répond avec un paquet ACK avertissant le serveur que ce dernier a reçu la réponse du serveur.

**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.

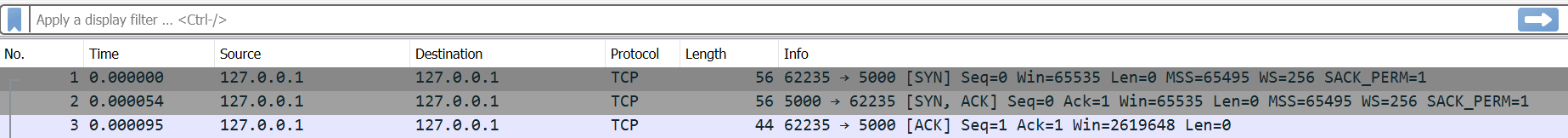


Figure 15: Three-way handshake du protocole TCP (Mode 2)

Cet échange est appelé le *Three-way handshake*. Il sert à synchroniser les numéros de séquence entre le client et le serveur afin d’établir la connexion entre ces derniers. Il débute par le client qui envoie un paquet SYN afin d’effectuer une demande de connexion avec le serveur. Ensuite, le serveur répond avec un paquet SYN-ACK signifiant qu’il accepte que le client se connecte. Finalement, le client répond avec un paquet ACK avertissant le serveur que ce dernier à reçu la réponse du serveur.

**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).

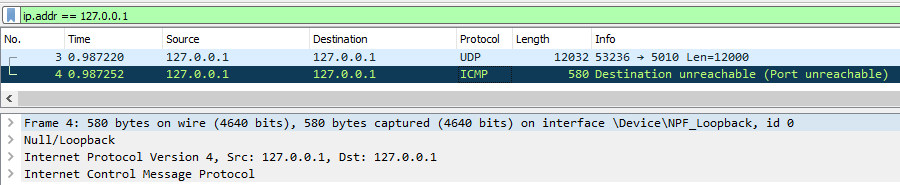


Figure 16: Absence de paquet ACK dans le protocole UDP (Mode 3)

**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 des paquets ICMP qui nous indiquent qu’il y a eu un problème de réseau empêchent la livraison du paquet (port inaccessible).

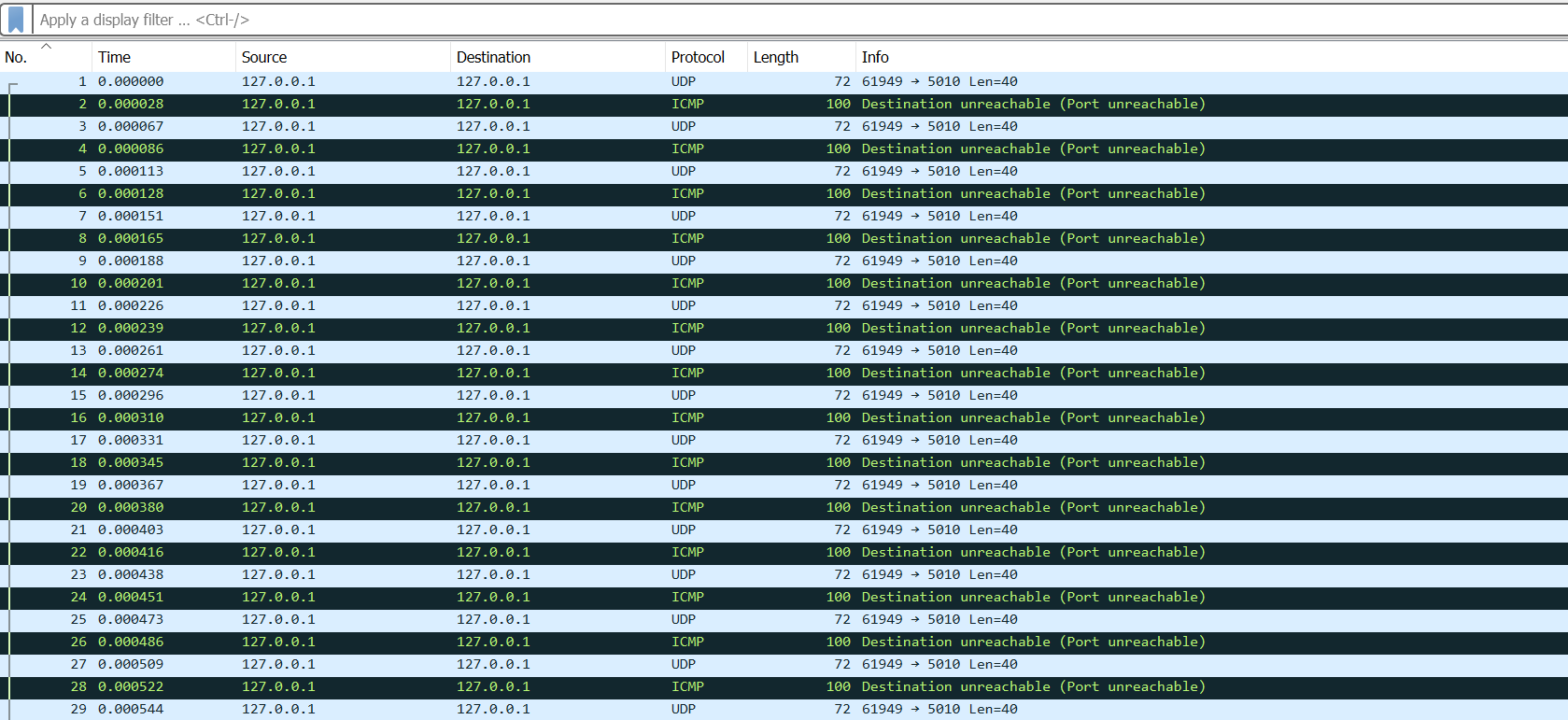


Figure 17: Absence de paquet ACK dans le protocole UDP (Mode 4)

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 comme on peut le voir dans la figure ci-dessous.

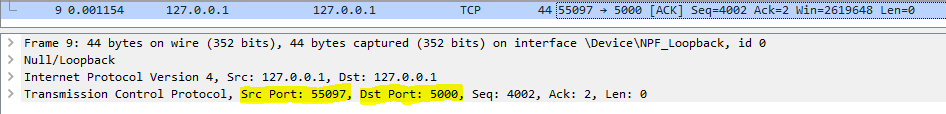


Figure 18: Identification du port source et du port destination (Mode 1)

**Mode 2**

Le port source est 62235 et le port de destination est 5000 comme on peut le voir dans la figure ci-dessous.

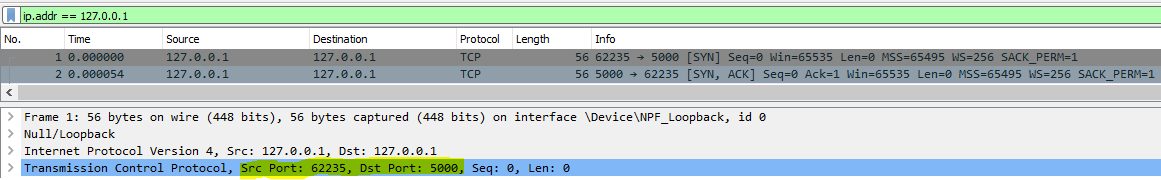


Figure 19: Identification du port source et du port destination (Mode 2)

**Mode 3**

Le port source est 53236 et le port de destination est 5010 comme on peut le voir dans la figure ci-dessous.

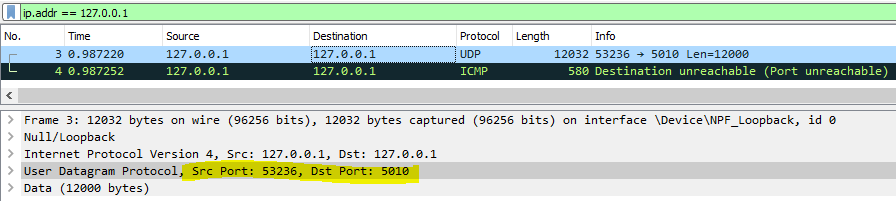


Figure 20: Identification du port source et du port destination (Mode 3)

**Mode 4**

Le port source est le 64393 et le port destination est le port 5010 comme on peut le voir dans la figure ci-dessous.

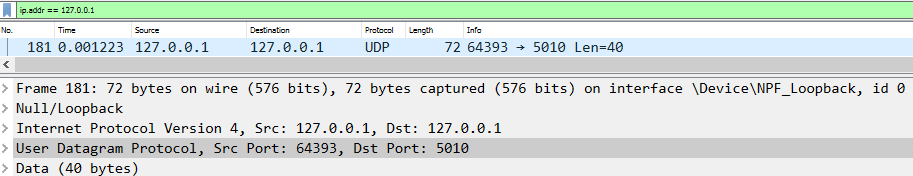




Figure 21: Identification du port source et du port destination (Mode 4)

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**

Nous pouvons voir, à l’aide de l’outil de statistiques de Wireshark et du filtre « ip.addr == 127.0.0.1 and tcp.srcport == 55097 and tcp.dstport == 5000 », que le client a envoyé 5 paquets au serveur.

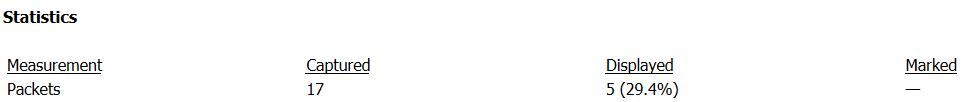




Figure 22: Affichage de l'outil de statistiques de Wireshark pour le nombre de paquets envoyés par le client

Ici, vu que le protocole employé est TCP, nous pouvons utiliser le numéro de séquence du dernier paquet envoyé par le client afin de déterminer le nombre d’octets de données envoyés au serveur. Selon la figure suivante, nous avons comme dernier numéro de séquence 4002.

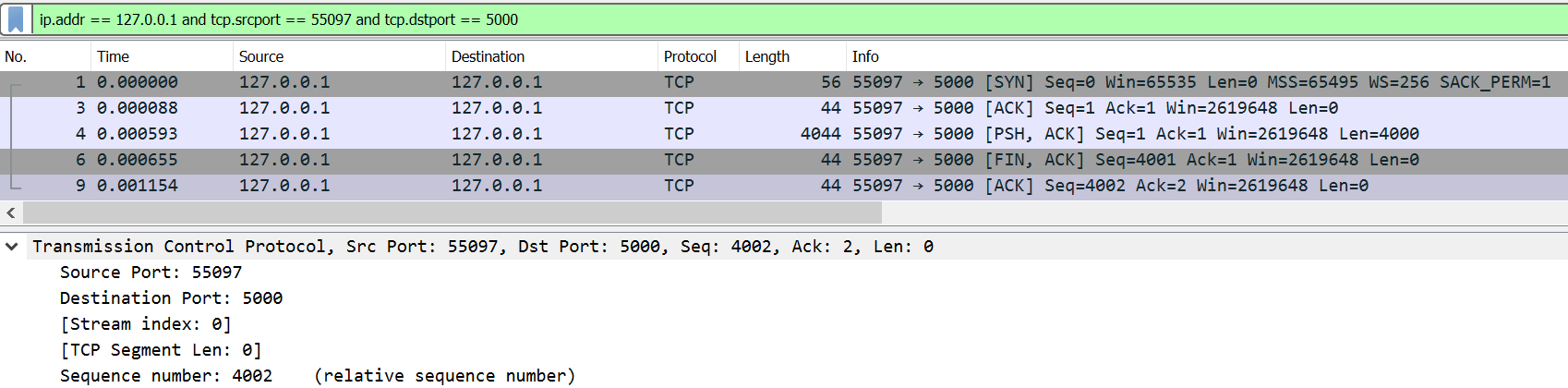




Figure 23: Numéro de séquence du dernier paquet envoyé par le client au serveur

Cependant, ce numéro de séquence est aussi incrémenté lors des échanges servant à initialiser la connexion et mettre fin à la connexion dans le cas du client puisque c’est lui qui a amorcé l’échange. Il faut alors soustraire 2 à ce nombre car, lors de l’initialisation et la fin de la connexion, aucune donnée « utile » qui est échangée entre le client et le serveur. Le client a alors envoyé 4000 octets de données au serveur.

Nous pouvons voir, à l’aide de l’outil de statistiques de Wireshark et du filtre « ip.addr == 127.0.0.1 and tcp.srcport == 5000 and tcp.dstport == 55097 », que le serveur a envoyé 4 paquets au client.





Figure 24: Affichage de l'outil de statistiques de Wireshark pour le nombre de paquets envoyés par le serveur

Ici, vu que le protocole employé est TCP, nous pouvons utiliser le numéro de séquence du dernier paquet envoyé par le serveur afin de déterminer le nombre d’octets de données envoyés au client. Selon la figure suivante, nous avons comme dernier numéro de séquence 1.

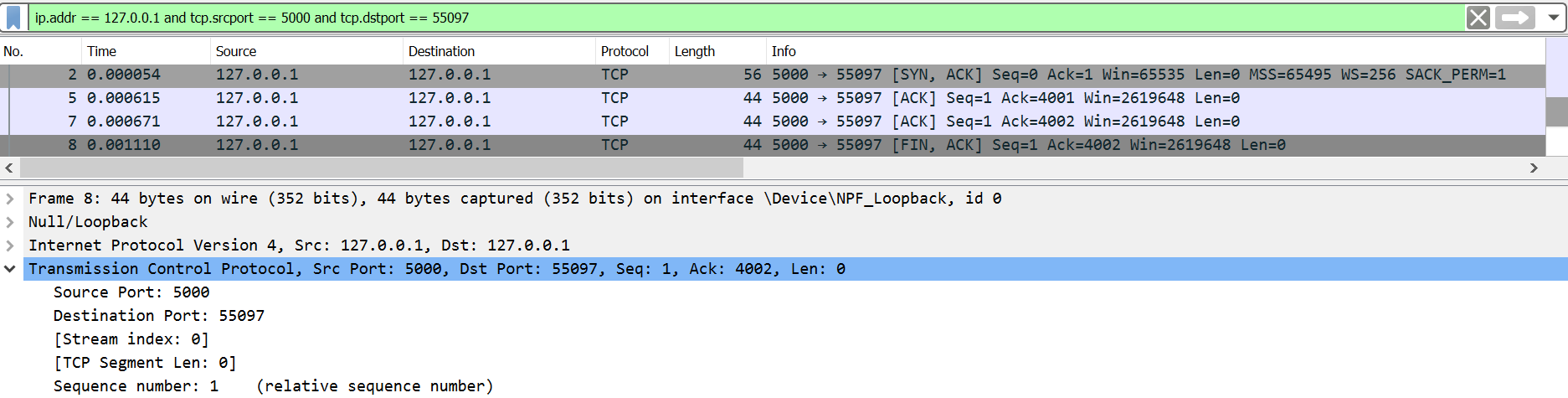




Figure 25: Numéro de séquence du dernier paquet envoyé par le serveur au client

Cependant, ce numéro de séquence est aussi incrémenté lors des échanges servant à initialiser la connexion le cas du serveur puisque c’est lui qui doit répondre à la demande de connexion du client. Il faut alors soustraire 1 à ce nombre car, lors de l’initialisation de la connexion, aucune donnée « utile » qui est échangée entre le client et le serveur. Le serveur n’a alors envoyé aucune donnée au client.

**Mode 2**

Nous pouvons voir, à l’aide de l’outil de statistiques de Wireshark et du filtre « ip.addr == 127.0.0.1 and tcp.srcport == 62235 and tcp.dstport == 5000 », que le client a envoyé 104 paquets au serveur.





Figure 26: Affichage de l'outil de statistiques de Wireshark pour le nombre de paquets envoyés par le client

Ici, vu que le protocole employé est TCP, nous pouvons utiliser le numéro de séquence du dernier paquet envoyé par le client afin de déterminer le nombre d’octets de données envoyés au serveur. Selon la figure suivante, nous avons comme dernier numéro de séquence 4002.

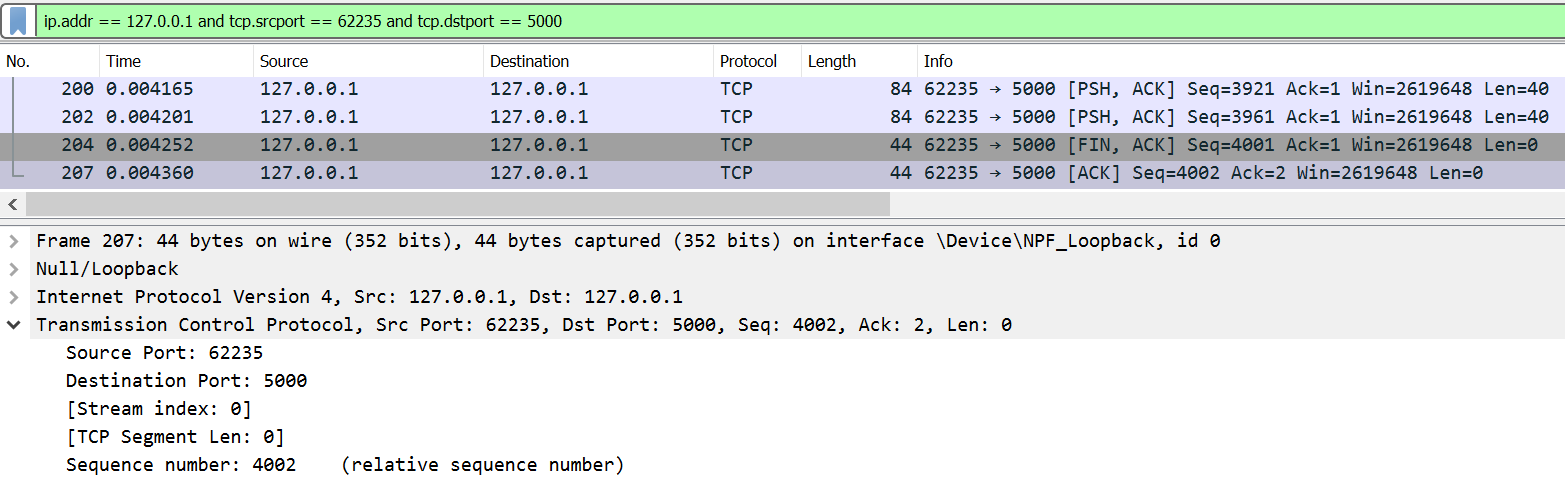




Figure 27: Numéro de séquence du dernier paquet envoyé par le client au serveur

Cependant, ce numéro de séquence est aussi incrémenté lors des échanges servant à initialiser la connexion et mettre fin à la connexion dans le cas du client puisque c’est lui qui a amorcé l’échange. Il faut alors soustraire 2 à ce nombre car, lors de l’initialisation et la fin de la connexion, aucune donnée « utile » qui est échangée entre le client et le serveur. Le client a alors envoyé 4000 octets de données au serveur.

Nous pouvons voir, à l’aide de l’outil de statistiques de Wireshark et du filtre « ip.addr == 127.0.0.1 and tcp.srcport == 5000 and tcp.dstport == 62235 », que le serveur a envoyé 103 paquets au client.





Figure 28: Affichage de l'outil de statistiques de Wireshark pour le nombre de paquets envoyés par le serveur

Ici, vu que le protocole employé est TCP, nous pouvons utiliser le numéro de séquence du dernier paquet envoyé par le serveur afin de déterminer le nombre d’octets de données envoyés au client.

Selon la figure suivante, nous avons comme dernier numéro de séquence 1.

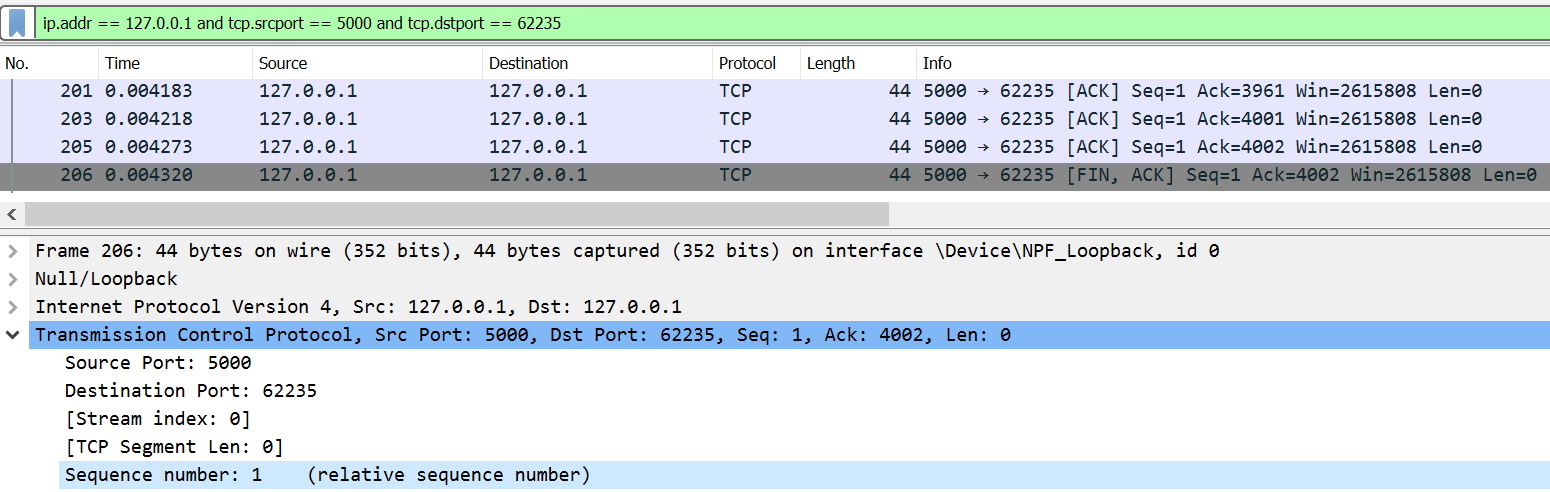




Figure 29: Numéro de séquence du dernier paquet envoyé par le serveur

Cependant, ce numéro de séquence est aussi incrémenté lors des échanges servant à initialiser la connexion le cas du serveur puisque c’est lui qui doit répondre à la demande de connexion du client. Il faut alors soustraire 1 à ce nombre car, lors de l’initialisation de la connexion, aucune donnée « utile » qui est échangée entre le client et le serveur. Le serveur n’a alors envoyé aucune donnée au client.

**Mode 3**

Le client a envoyé 1 paquet UDP contenant 12000 octets de données (voir la figure ci-dessous).

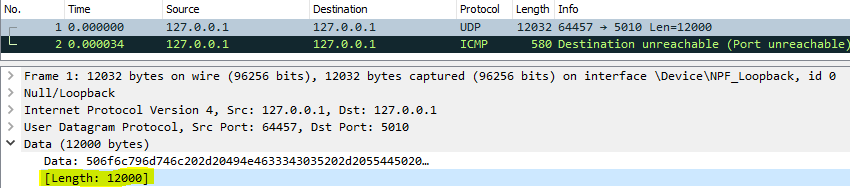


Figure 30: Longueur du paquet envoyé par le client

Le serveur n’a envoyé aucun paquet au client.

**Mode 4**

Le client a envoyé 300 paquets UDP identiques contenant tous 40 octets de données tel qu’indiqué dans la figure ci-dessous.

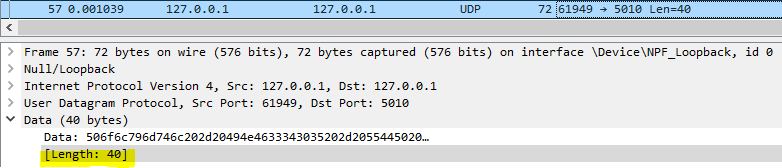


Figure 31: Longueur des paquets envoyés par le client

Nous pouvons alors calculer le nombre total d’octets de données envoyés par le client de la façon suivante :

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 4000 octets de données, ce qui est plus grand qu’un paquet « habituel ». Nous pensons que le client a fait 3 itérations pour envoyer la totalité de ses données, car le numéro de séquence a été incrémenté 3 fois. Ainsi, nous pensons que le client a fait une première itération pour envoyer un seul octet de donnée, puis une deuxième itération pour envoyer un gros morceau de l’information et une troisième itération pour envoyer un seul octet d’information.

**Mode 2**

Dans ce mode, le client envoi la même quantité d’informations au serveur vu le dernier numéro de séquence atteint (4002). Cependant, il le fait en 103 itérations, car le numéro de séquence a seulement incrémenté sur 103 paquets.

**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. Le client a alors fait 300 itérations pour envoyer ses 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)**

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. De plus, le nombre d’itérations diffère entre les deux modes. Ainsi, le mode 1 est beaucoup plus performant que le mode 2, car il envoi la même quantité de données plus rapidement et avec un plus petit nombre d’itérations.

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.

De plus, le nombre d’itération qui diffère ici aussi. Clairement, le mode 3 semble plus performant que le mode 4 à ce niveau vu qu’il envoi toutes ses données en une seule itération. Cependant, vu que le protocole est UDP, le mode 3 serait moins performant dans d’autres applications. En effet, les applications employant le protocole UDP font souvent des opérations en temps réel et il serait alors plus performant de manquer quelques paquets d’un ensemble d’information plutôt que de devoir renvoyer l’ensemble au complet jusqu’à ce que le destinataire réussisse à le recevoir. Ainsi, le mode 4 serait probablement le plus performant en raison des applications exigeant des opérations en temps réel.

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. Ainsi, si un paquet n’est pas reçu par le destinataire, 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 serveur confirme au client la réception de ce paquet (avec un paquet ACK). 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.