

École Polytechnique de Montréal



INF3405
Réseaux informatiques

Hiver 2020 laboratoire 3

Soumis par
Roman Zhornytskiy (1899786) et
Hakim Payman (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'adresse 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.

ip.addr == 127.0.0.1 and tcp.port eq 5000									
No.	Time	Source	Destination	Protocol	Length	Info			
75	12.263370	127.0.0.1	127.0.0.1	TCP	56	59698 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1			
76	12.263423	127.0.0.1	127.0.0.1	TCP	56	5000 → 59698 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1			
77	12.263464	127.0.0.1	127.0.0.1	TCP	44	59698 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0			
78	12.264578	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet: length of contained item exceeds length of containing item]			
79	12.264604	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=5 Win=2619648 Len=0			
80	12.269810	127.0.0.1	127.0.0.1	RSL	85	unknown 106			
81	12.269836	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=46 Win=2619648 Len=0			
82	12.270040	127.0.0.1	127.0.0.1	TCP	46	59698 → 5000 [PSH, ACK] Seq=46 Ack=1 Win=2619648 Len=2			
83	12.270062	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=48 Win=2619648 Len=0			
84	12.270130	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet]			
85	12.270148	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=52 Win=2619648 Len=0			
86	12.270189	127.0.0.1	127.0.0.1	TCP	46	59698 → 5000 [PSH, ACK] Seq=52 Ack=1 Win=2619648 Len=2			
87	12.270207	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=54 Win=2619648 Len=0			
88	12.270228	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet]			
89	12.270245	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=58 Win=2619648 Len=0			
90	12.271034	127.0.0.1	127.0.0.1	RSL	52	unknown 111			
91	12.271055	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=66 Win=2619648 Len=0			
92	12.271096	127.0.0.1	127.0.0.1	RSL	50	PREPROCESSED MEASUREMENT RESULT [Malformed Packet: length of contained item exceeds length of containing item]			
93	12.271114	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=72 Win=2619648 Len=0			
94	12.271143	127.0.0.1	127.0.0.1	TCP	45	59698 → 5000 [PSH, ACK] Seq=72 Ack=1 Win=2619648 Len=1			
95	12.271160	127.0.0.1	127.0.0.1	TCP	44	5000 → 59698 [ACK] Seq=1 Ack=73 Win=2619648 Len=0			
96	12.271527	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet: length of contained item exceeds length of containing item]			
97	12.271550	127.0.0.1	127.0.0.1	TCP	44	59698 → 5000 [ACK] Seq=73 Ack=5 Win=2619648 Len=0			
98	12.271595	127.0.0.1	127.0.0.1	RSL	84	unknown 106			
99	12.271613	127.0.0.1	127.0.0.1	TCP	44	59698 → 5000 [ACK] Seq=73 Ack=45 Win=2619648 Len=0			
100	12.271640	127.0.0.1	127.0.0.1	TCP	46	5000 → 59698 [PSH, ACK] Seq=45 Ack=73 Win=2619648 Len=2			
101	12.271656	127.0.0.1	127.0.0.1	TCP	44	59698 → 5000 [ACK] Seq=73 Ack=47 Win=2619648 Len=0			
102	12.271675	127.0.0.1	127.0.0.1	TCP	45	5000 → 59698 [PSH, ACK] Seq=47 Ack=73 Win=2619648 Len=1			
103	12.271691	127.0.0.1	127.0.0.1	TCP	44	59698 → 5000 [ACK] Seq=73 Ack=48 Win=2619648 Len=0			
137	21.238901	127.0.0.1	127.0.0.1	RSL	60	unknown 111			

> Frame 87: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{...} id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 5000, Dst Port: 59698, Seq: 1, Ack: 54, Len: 0

0000	02 00 00 00 45 00 00 28	ac 10 40 00 80 06 00 00E..(.@....
0010	7f 00 00 01 7f 00 00 01	13 88 e9 32 6b 8d 96 cc2k...
0020	9a 5c a1 fc 50 10 27 f9	4e 6b 00 00	...\P...NK..

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 :

ip.addr == 127.0.0.1 and tcp.dstport == 58143 and tcp.srcport == 5000						
No.	Time	Source	Destination	Protocol	Length	Info
2	0.000038	127.0.0.1	127.0.0.1	TCP	56	5000 → 58143 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	0.004151	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
7	0.012412	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=6 Win=2619648 Len=0
9	0.012750	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=48 Win=2619648 Len=0
11	0.012838	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=52 Win=2619648 Len=0
13	0.012873	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=54 Win=2619648 Len=0
15	0.012893	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=58 Win=2619648 Len=0
17	0.018387	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=63 Win=2619648 Len=0
19	0.018429	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=68 Win=2619648 Len=0
21	0.018455	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=1 Ack=69 Win=2619648 Len=0
22	0.020591	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet: length of contained item exceeds length of containing item]
24	0.023378	127.0.0.1	127.0.0.1	RSL	84	unknown 106
26	0.023450	127.0.0.1	127.0.0.1	TCP	46	5000 → 58143 [PSH, ACK] Seq=45 Ack=69 Win=2619648 Len=2
28	0.023472	127.0.0.1	127.0.0.1	TCP	45	5000 → 58143 [PSH, ACK] Seq=47 Ack=69 Win=2619648 Len=1
35	9.398757	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=48 Ack=85 Win=2619648 Len=0
37	9.971389	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=48 Ack=89 Win=2619648 Len=0
39	9.971519	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=48 Ack=11451 Win=2608128 Len=0
40	10.214733	127.0.0.1	127.0.0.1	TCP	48	5000 → 58143 [PSH, ACK] Seq=48 Ack=11451 Win=2608128 Len=4
42	10.214819	127.0.0.1	127.0.0.1	OML	16682	OML
44	10.215601	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [FIN, ACK] Seq=16690 Ack=11451 Win=2608128 Len=0
47	10.223051	127.0.0.1	127.0.0.1	TCP	44	5000 → 58143 [ACK] Seq=16691 Ack=11452 Win=2608128 Len=0

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

Statistics

Measurement	Captured	Displayed	Marked
Packets	47	21 (44.7%)	—
Time span, s	10.223	10.223	—
Average pps	4.6	2.1	—
Average packet size, B	657	839	—
Bytes	30879	17625 (57.1%)	0
Average bytes/s	3020	1724	—
Average bits/s	24 k	13 k	—

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 :

ip.addr == 127.0.0.1 and tcp.dstport == 5000 and tcp.srcport == 58143						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	58143 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000087	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.004134	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet: length of contained item exceeds length of containing item]
6	0.012391	127.0.0.1	127.0.0.1	RSL	85	unknown 106
8	0.012740	127.0.0.1	127.0.0.1	TCP	46	58143 → 5000 [PSH, ACK] Seq=46 Ack=1 Win=2619648 Len=2
10	0.012831	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet]
12	0.012867	127.0.0.1	127.0.0.1	TCP	46	58143 → 5000 [PSH, ACK] Seq=52 Ack=1 Win=2619648 Len=2
14	0.012888	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet]
16	0.018352	127.0.0.1	127.0.0.1	RSL	49	unknown 111
18	0.018423	127.0.0.1	127.0.0.1	RSL	49	unknown 111
20	0.018449	127.0.0.1	127.0.0.1	TCP	45	58143 → 5000 [PSH, ACK] Seq=68 Ack=1 Win=2619648 Len=1
23	0.020609	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=69 Ack=5 Win=2619648 Len=0
25	0.023392	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=69 Ack=45 Win=2619648 Len=0
27	0.023457	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=69 Ack=47 Win=2619648 Len=0
29	0.023477	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=69 Ack=48 Win=2619648 Len=0
34	9.398709	127.0.0.1	127.0.0.1	RSL	60	unknown 111
36	9.971363	127.0.0.1	127.0.0.1	TCP	48	58143 → 5000 [PSH, ACK] Seq=85 Ack=48 Win=2619648 Len=4
38	9.971502	127.0.0.1	127.0.0.1	OML	11406	OML
41	10.214759	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=11451 Ack=52 Win=2619648 Len=0
43	10.214831	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=11451 Ack=16690 Win=2603008 Len=0
45	10.215617	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [ACK] Seq=11451 Ack=16691 Win=2603008 Len=0
46	10.223013	127.0.0.1	127.0.0.1	TCP	44	58143 → 5000 [FIN, ACK] Seq=11451 Ack=16691 Win=2603008 Len=0

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

Statistics

Measurement	Captured	Displayed	Marked
Packets	47	22 (46.8%)	—
Time span, s	10.223	10.223	—
Average pps	4.6	2.2	—
Average packet size, B	657	565	—
Bytes	30879	12430 (40.3%)	0
Average bytes/s	3020	1215	—
Average bits/s	24 k	9727	—

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.

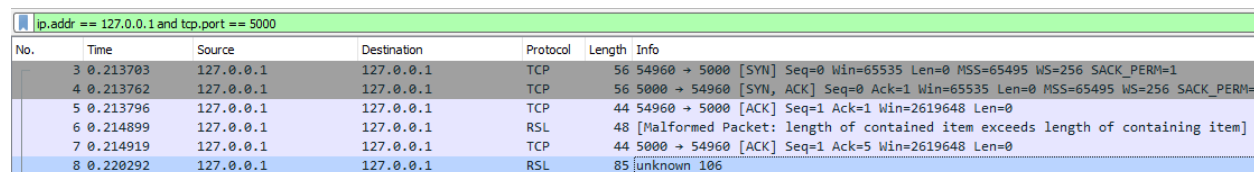
4) Normalement, le standard IEEE 802.3 limite la taille d'une trame *Ethernet* à 1518 octets. Dans votre capture Wireshark, existe-t-il des paquets ayant une taille supérieure à 1518 octets? Si oui, expliquez pourquoi et comment ce paquet réussit à transiger sur le réseau alors que sa taille est plus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



No.	Time	Source	Destination	Protocol	Length	Info
3	0.213703	127.0.0.1	127.0.0.1	TCP	56	54960 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
4	0.213762	127.0.0.1	127.0.0.1	TCP	56	5000 → 54960 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	0.213796	127.0.0.1	127.0.0.1	TCP	44	54960 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
6	0.214899	127.0.0.1	127.0.0.1	RSL	48	[Malformed Packet: length of contained item exceeds length of containing item]
7	0.214919	127.0.0.1	127.0.0.1	TCP	44	5000 → 54960 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
8	0.220292	127.0.0.1	127.0.0.1	RSL	85	unknown 106

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.

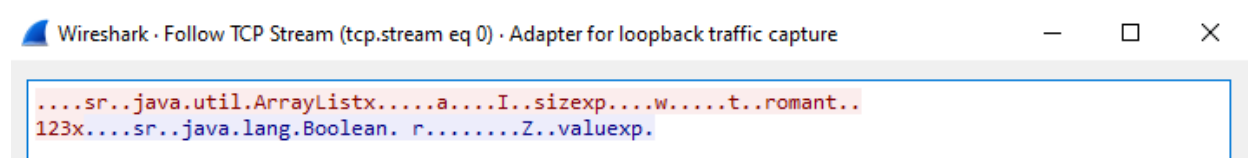


Figure 7: Nom d'utilisateur et mot de passe dans un paquet

Dans cette figure, on peut observer le nom d'utilisateur de l'utilisateur qui est « roman » et quelques caractères 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.

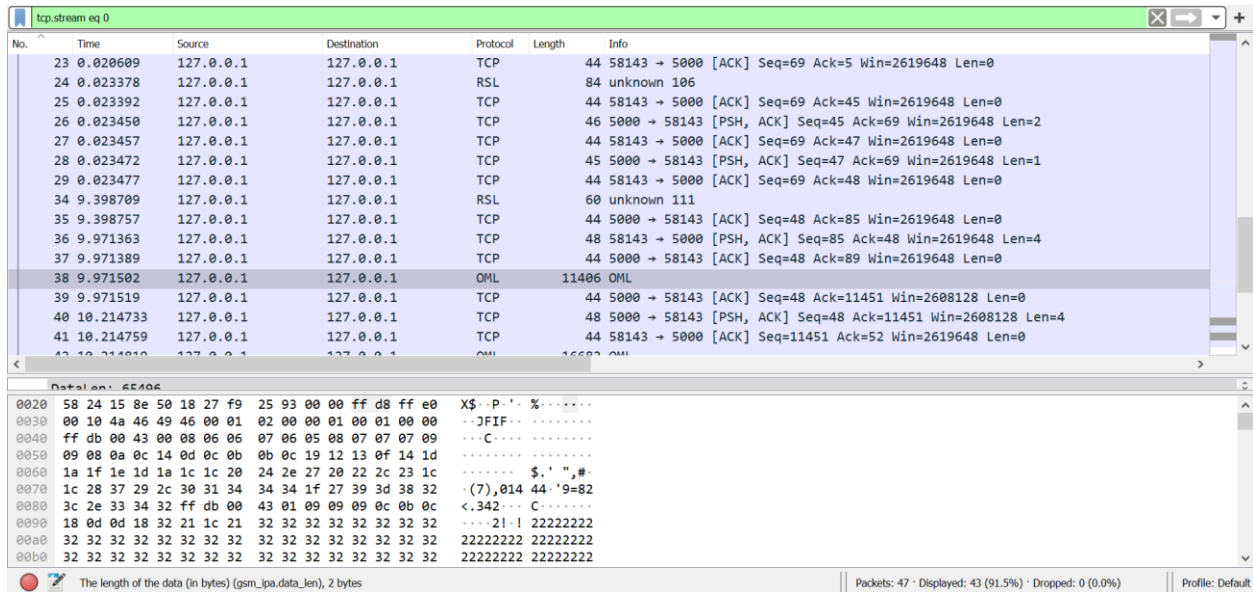


Figure 8: Paquet contenant l'image

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

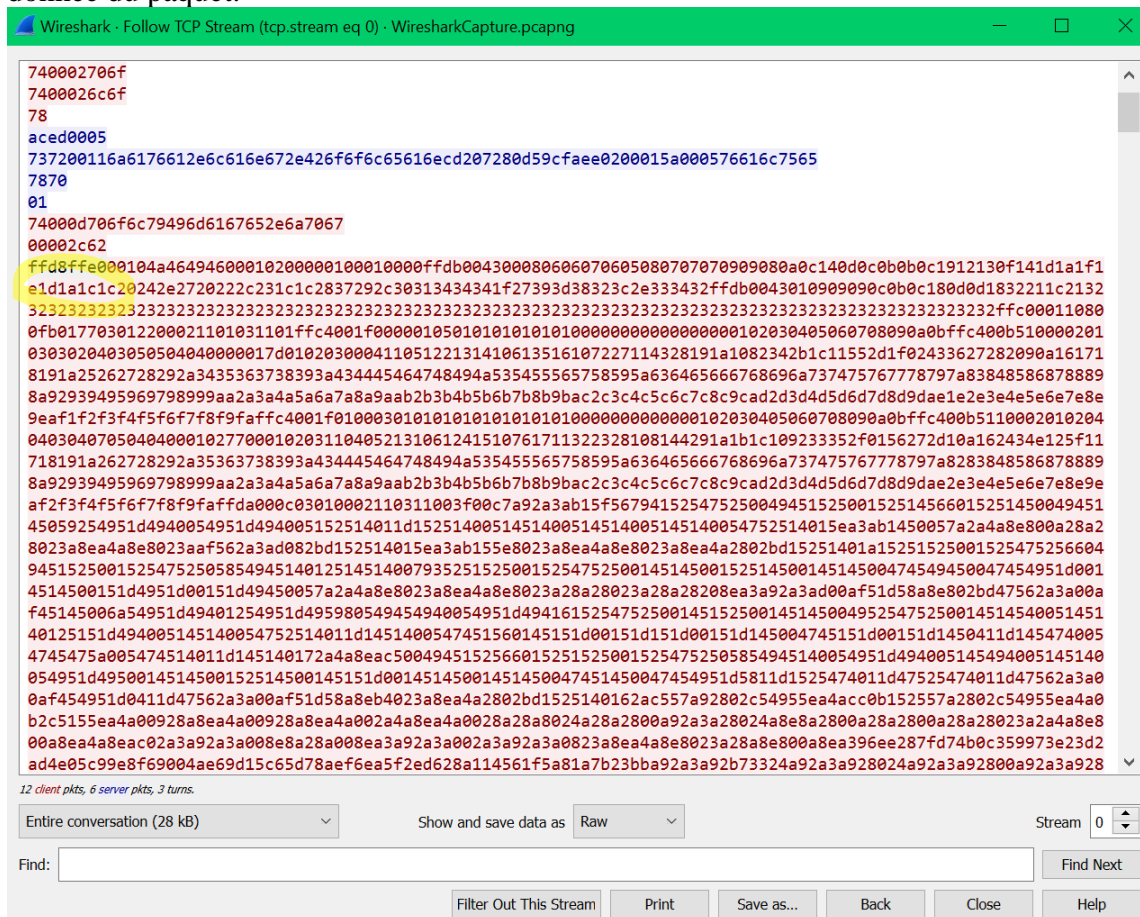


Figure 9: Contenu du paquet contenant l'image

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.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	AC	ED	00	05	73	72	00	13	6A	61	76	61	2E	75	74	69	~í sr java.util
00000010	6C	2E	41	72	72	61	79	4C	69	73	74	78	81	D2	1D	99	l.ArrayListx ò ¢
00000020	C7	61	9D	03	00	01	49	00	04	73	69	7A	65	78	70	00	Ça I sizexp
00000030	00	00	02	77	04	00	00	00	02	74	00	02	70	6F	74	00	w t pot
00000040	02	6C	6F	78	AC	ED	00	05	73	72	00	11	6A	61	76	61	lox~í sr java
00000050	2E	6C	61	6E	67	2E	42	6F	6F	6C	65	61	6E	CD	20	72	.lang.Booleaní r
00000060	80	D5	9C	FA	EE	02	00	01	5A	00	05	76	61	6C	75	65	€Œœuí Z value
00000070	78	70	01	74	00	0D	70	6F	6C	79	49	6D	61	67	65	2E	xp t polyImage.
00000080	6A	70	67	00	00	2C	62	FF	D8	FF	E0	00	10	4A	46	49	jpg ,bÿøÿà JFIF
00000090	46	00	01	02	00	00	01	00	01	00	00	FF	DB	00	43	00	F ÿÛ C
000000A0	08	06	06	07	06	05	08	07	07	07	09	09	08	0A	0C	14	
000000B0	0D	0C	0B	0B	0C	19	12	13	0F	14	1D	1A	1F	1E	1D	1A	
000000C0	1C	1C	20	24	2E	27	20	22	2C	23	1C	1C	28	37	29	2C	\$. ' ",# (7),
000000D0	30	31	34	34	34	1F	27	39	3D	38	32	3C	2E	33	34	32	01444 '9=82<.342
000000E0	FF	DB	00	43	01	09	09	09	0C	0B	0C	18	0D	0D	18	32	ÿÛ C 2
000000F0	21	1C	21	32	32	32	32	32	32	32	32	32	32	32	32	32	! !222222222222
00000100	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	22222222222222
00000110	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	22222222222222
00000120	32	32	32	32	32	FF	C0	00	11	08	00	FB	01	77	03	01	22222ÿÀ û w
00000130	22	00	02	11	01	03	11	01	FF	C4	00	1F	00	00	01	05	" ÿÀ

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

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	02	00	00	01	ÿøÿà JFIF
00000010	00	01	00	00	FF	DB	00	43	00	08	06	06	07	06	05	08	ÿÛ C
00000020	07	07	07	09	09	08	0A	0C	14	0D	0C	0B	0B	0C	19	12	
00000030	13	0F	14	1D	1A	1F	1E	1D	1A	1C	1C	20	24	2E	27	20	
00000040	22	2C	23	1C	1C	28	37	29	2C	30	31	34	34	34	1F	27	\$. ' ",# (7),01444 '
00000050	39	3D	38	32	3C	2E	33	34	32	FF	DB	00	43	01	09	09	9=82<.342ÿÛ C
00000060	09	0C	0B	0C	18	0D	0D	18	32	21	1C	21	32	32	32	32	2! !2222
00000070	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	22222222222222
00000080	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	22222222222222
00000090	32	32	32	32	32	32	32	32	32	32	32	32	32	32	FF	C0	2222222222222ÿÀ
000000A0	00	11	08	00	FB	01	77	03	01	22	00	02	11	01	03	11	û w "
000000B0	01	FF	C4	00	1F	00	00	01	05	01	01	01	01	01	01	00	ÿÀ
000000C0	00	00	00	00	00	00	00	01	02	03	04	05	06	07	08	09	
000000D0	0A	0B	FF	C4	00	B5	10	00	02	01	03	03	02	04	03	05	ÿÀ µ
000000E0	05	04	04	00	00	01	7D	01	02	03	00	04	11	05	12	21	} !

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

00006C70	ED 32 60 3A ED 61 61 C6	A0 0E 0E CA 39 24 60 37	1 : :1a E 4E39 -
00006C80	0C B3 43 27 F1 BA 9C 0E	A7 38 1D C5 07 EC D0 2C	'C'ñ°œ \$8 Å ið,
00006C90	8C 10 38 DC 76 16 E0 28	C7 56 03 F4 19 AE 7E 3D	œ 8Üv à(ÇV ô œ~=
00006CA0	46 F2 32 0A 5C 38 C6 78	CF 1C F0 78 E9 DE A3 37	Fò2 \8ExÍ ôxéBf7
00006CB0	53 15 2A 5F 83 D4 6D 14	01 A7 2D CA B2 05 8D 58	S *_fôm S-Ê² X
00006CC0	29 EA D8 C3 31 EF CF F4	ED 56 B4 D4 8D 10 4D 11)êØÃ1iïôiv'ô M
00006CD0	08 C0 7C CE 3E 67 FC 17	B7 D6 B1 16 FA E5 13 62	À î>gü ·Ö± úå b
00006CE0	C9 85 C6 3E E8 E9 FE 45	39 35 1B B8 C6 12 76 55	É...E>èépE95 ,Æ vU
00006CF0	EB 85 C0 07 EA 3B D0 06	D5 F5 C0 00 95 07 6E 31	ë..À ê;Ð ÔçÀ · n1
00006D00	B5 9B 3E 9C 9F EF 1C FE	5C 56 4B A1 47 CA AE 03	µ>>œÿi p\VK;GÊ®
00006D10	7C C5 7D 07 A5 55 6B 89	58 E4 B9 34 86 69 09 24	Å} ¥UkXä'4ti \$
00006D20	B7 27 AD 00 68 27 92 32	24 45 5F 43 92 47 E9 50	·'- h''2\$E_C'GéP
00006D30	97 05 88 03 3F F0 1E B5	50 BB 13 92 68 0E C0 E4	- ^ ?ð µP» 'h Àä
00006D40	1C 1A 00 9C C1 30 B6 92	43 0C 81 01 19 62 A7 03	αÁ0¶'C bS
00006D50	9F 5A 29 AD 79 70 D0 18	5A 67 31 93 92 B9 E0 D1	ÝZ)-ypÐ Zg1''¹àÑ
00006D60	40 1F FF D9		@ ŷÛ

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.

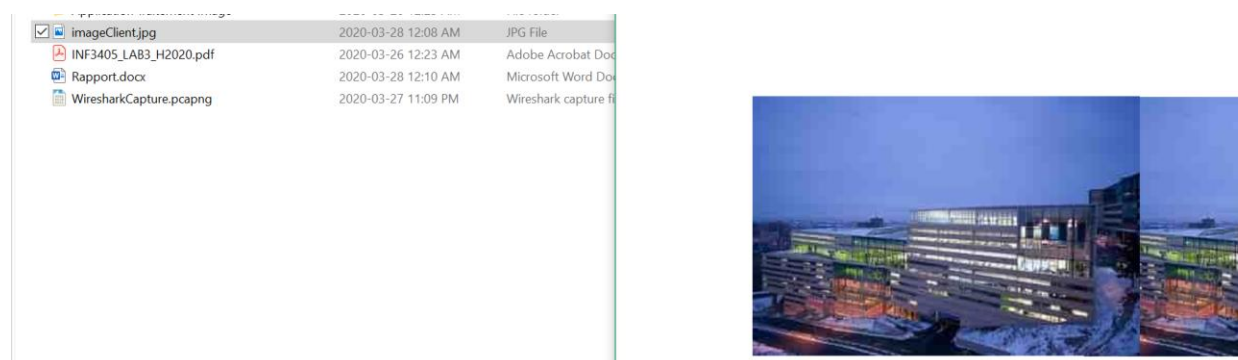


Figure 13: Résultat de l'ouverture en .jpg de l'image extraite d'un paquet

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)

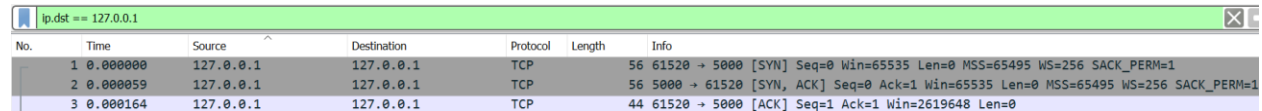
L'application de procure aucune sécurité en ce a trait aux images et aux données qui sont envoyées. En effet, on a pu très facilement extraire l'image envoyé de son paquet seulement en sachant le format de l'image et en utilisant des outils faciles à obtenir (Winhex et Wireshark). Il en va de même pour 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 a 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.



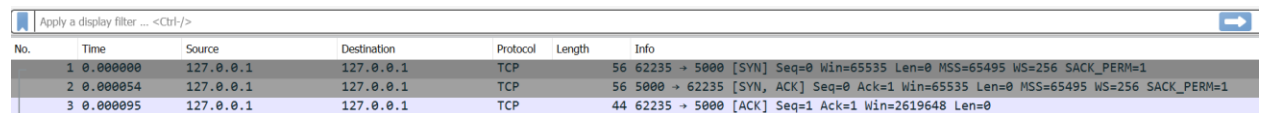
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	61520 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000059	127.0.0.1	127.0.0.1	TCP	56	5000 → 61520 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000164	127.0.0.1	127.0.0.1	TCP	44	61520 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0

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.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	62235 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000054	127.0.0.1	127.0.0.1	TCP	56	5000 → 62235 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000095	127.0.0.1	127.0.0.1	TCP	44	62235 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0

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 a 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 empêchant la livraison du paquet (port inaccessible).

ip.addr == 127.0.0.1						
No.	Time	Source	Destination	Protocol	Length	Info
3	0.987220	127.0.0.1	127.0.0.1	UDP	12032	53236 → 5010 Len=12000
4	0.987252	127.0.0.1	127.0.0.1	ICMP	580	Destination unreachable (Port unreachable)

<

> Frame 4: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface \Device\NPF_{Loopback}, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Internet Control Message Protocol

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 empêchant la livraison du paquet (port inaccessible).

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
2	0.000028	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
3	0.000067	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
4	0.000086	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
5	0.000113	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
6	0.000128	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
7	0.000151	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
8	0.000165	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
9	0.000188	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
10	0.000201	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
11	0.000226	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
12	0.000239	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
13	0.000261	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
14	0.000274	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
15	0.000296	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
16	0.000310	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
17	0.000331	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
18	0.000345	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
19	0.000367	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
20	0.000380	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
21	0.000403	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
22	0.000416	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
23	0.000438	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
24	0.000451	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
25	0.000473	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
26	0.000486	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
27	0.000509	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
28	0.000522	127.0.0.1	127.0.0.1	ICMP	100	Destination unreachable (Port unreachable)
29	0.000544	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40

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.

9	0.001154	127.0.0.1	127.0.0.1	TCP	44	55097 → 5000 [ACK] Seq=4002 Ack=2 Win=2619648 Len=0
---	----------	-----------	-----------	-----	----	---

> Frame 9: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0

> Null/Loopback

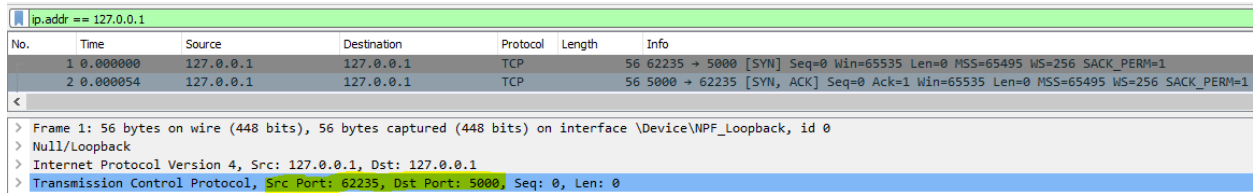
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 55097, Dst Port: 5000, Seq: 4002, Ack: 2, Len: 0

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.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	62235 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000054	127.0.0.1	127.0.0.1	TCP	56	5000 → 62235 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1

> Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{Loopback}, id 0

> Null/Loopback

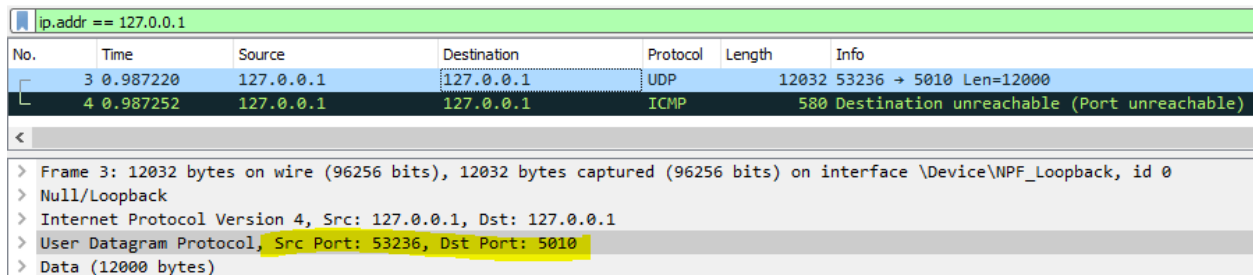
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 62235, Dst Port: 5000, Seq: 0, Len: 0

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.



No.	Time	Source	Destination	Protocol	Length	Info
3	0.987220	127.0.0.1	127.0.0.1	UDP	12032	53236 → 5010 Len=12000
4	0.987252	127.0.0.1	127.0.0.1	ICMP	580	Destination unreachable (Port unreachable)

> Frame 3: 12032 bytes on wire (96256 bits), 12032 bytes captured (96256 bits) on interface \Device\NPF_{Loopback}, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

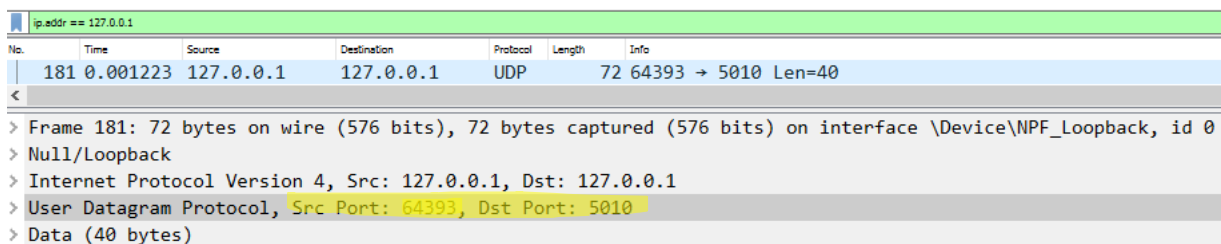
> User Datagram Protocol, Src Port: 53236, Dst Port: 5010

> Data (12000 bytes)

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.



No.	Time	Source	Destination	Protocol	Length	Info
181	0.001223	127.0.0.1	127.0.0.1	UDP	72	64393 → 5010 Len=40

> Frame 181: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface \Device\NPF_{Loopback}, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> User Datagram Protocol, Src Port: 64393, Dst Port: 5010

> Data (40 bytes)

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.srport == 55097 and tcp.dstport == 5000 », que le client a envoyé 5 paquets au serveur.

Statistics

Measurement	Captured	Displayed	Marked
Packets	17	5 (29.4%)	—

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.

ip.addr == 127.0.0.1 and tcp.srcport == 55097 and tcp.dstport == 5000						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	55097 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000088	127.0.0.1	127.0.0.1	TCP	44	55097 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.000593	127.0.0.1	127.0.0.1	TCP	4044	55097 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=4000
6	0.000655	127.0.0.1	127.0.0.1	TCP	44	55097 → 5000 [FIN, ACK] Seq=4001 Ack=1 Win=2619648 Len=0
9	0.001154	127.0.0.1	127.0.0.1	TCP	44	55097 → 5000 [ACK] Seq=4002 Ack=2 Win=2619648 Len=0

Transmission Control Protocol, Src Port: 55097, Dst Port: 5000, Seq: 4002, Ack: 2, Len: 0
Source Port: 55097
Destination Port: 5000
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 4002 (relative sequence number)

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.

Statistics

Measurement	Captured	Displayed	Marked
Packets	17	4 (23.5%)	—

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.

ip.addr == 127.0.0.1 and tcp.srcport == 5000 and tcp.dstport == 55097						
No.	Time	Source	Destination	Protocol	Length	Info
2	0.000054	127.0.0.1	127.0.0.1	TCP	56	5000 → 55097 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	0.000615	127.0.0.1	127.0.0.1	TCP	44	5000 → 55097 [ACK] Seq=1 Ack=4001 Win=2619648 Len=0
7	0.000671	127.0.0.1	127.0.0.1	TCP	44	5000 → 55097 [ACK] Seq=1 Ack=4002 Win=2619648 Len=0
8	0.001110	127.0.0.1	127.0.0.1	TCP	44	5000 → 55097 [FIN, ACK] Seq=1 Ack=4002 Win=2619648 Len=0

Frame 8: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 5000, Dst Port: 55097, Seq: 1, Ack: 4002, Len: 0
Source Port: 5000
Destination Port: 55097
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1 (relative sequence number)

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.

Statistics

<u>Measurement</u>	<u>Captured</u>	<u>Displayed</u>	<u>Marked</u>
Packets	207	104 (50.2%)	—

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.

ip.addr == 127.0.0.1 and tcp.srcport == 62235 and tcp.dstport == 5000							
No.	Time	Source	Destination	Protocol	Length	Info	
200	0.004165	127.0.0.1	127.0.0.1	TCP	84	62235 → 5000 [PSH, ACK] Seq=3921 Ack=1 Win=2619648 Len=40	
202	0.004201	127.0.0.1	127.0.0.1	TCP	84	62235 → 5000 [PSH, ACK] Seq=3961 Ack=1 Win=2619648 Len=40	
204	0.004252	127.0.0.1	127.0.0.1	TCP	44	62235 → 5000 [FIN, ACK] Seq=4001 Ack=1 Win=2619648 Len=0	
207	0.004360	127.0.0.1	127.0.0.1	TCP	44	62235 → 5000 [ACK] Seq=4002 Ack=2 Win=2619648 Len=0	

> Frame 207: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0 > Null/Loopback > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 ✓ Transmission Control Protocol, Src Port: 62235, Dst Port: 5000, Seq: 4002, Ack: 2, Len: 0 Source Port: 62235 Destination Port: 5000 [Stream index: 0] [TCP Segment Len: 0] Sequence number: 4002 (relative sequence number)							
--	--	--	--	--	--	--	--

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.

Statistics

<u>Measurement</u>	<u>Captured</u>	<u>Displayed</u>	<u>Marked</u>
Packets	207	103 (49.8%)	—

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.

ip.addr == 127.0.0.1 and tcp.srcport == 5000 and tcp.dstport == 62235						
No.	Time	Source	Destination	Protocol	Length	Info
201	0.004183	127.0.0.1	127.0.0.1	TCP	44	5000 → 62235 [ACK] Seq=1 Ack=3961 Win=2615808 Len=0
203	0.004218	127.0.0.1	127.0.0.1	TCP	44	5000 → 62235 [ACK] Seq=1 Ack=4001 Win=2615808 Len=0
205	0.004273	127.0.0.1	127.0.0.1	TCP	44	5000 → 62235 [ACK] Seq=1 Ack=4002 Win=2615808 Len=0
206	0.004320	127.0.0.1	127.0.0.1	TCP	44	5000 → 62235 [FIN, ACK] Seq=1 Ack=4002 Win=2615808 Len=0


```

> Frame 206: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 5000, Dst Port: 62235, Seq: 1, Ack: 4002, Len: 0
  Source Port: 5000
  Destination Port: 62235
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)

```

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).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	12032	64457 → 5010 Len=12000
2	0.000034	127.0.0.1	127.0.0.1	ICMP	580	Destination unreachable (Port unreachable)


```

> Frame 1: 12032 bytes on wire (96256 bits), 12032 bytes captured (96256 bits) on interface \Device\NPF_{Loopback}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 64457, Dst Port: 5010
> Data (12000 bytes)
  Data: 506f6c796d746c202d20494e4633343035202d2055445020...
  [Length: 12000]

```

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.

57	0.001039	127.0.0.1	127.0.0.1	UDP	72	61949 → 5010 Len=40
----	----------	-----------	-----------	-----	----	---------------------


```

> Frame 57: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface \Device\NPF_{Loopback}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 61949, Dst Port: 5010
> Data (40 bytes)
  Data: 506f6c796d746c202d20494e4633343035202d2055445020...
  [Length: 40]

```

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 :

$$\left(300 \text{ paquets} \times 40 \frac{\text{octets}}{\text{paquets}}\right) = 12000 \text{ octets}$$

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 envoie 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 fichier. 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 envoie 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 envoie 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 envoyer 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.