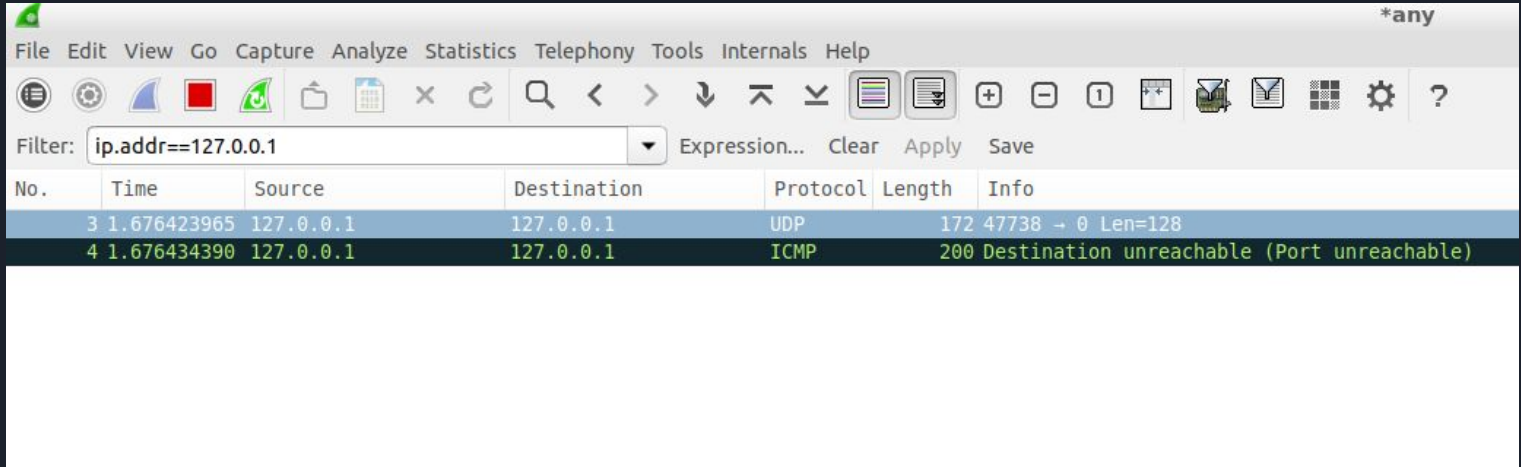




# Prog Sys : TP2

## Wireshark

# Tentative d'envoi de paquet en UDP



The image shows a Wireshark network traffic capture. The filter bar at the top is set to 'ip.addr==127.0.0.1'. The packet list shows two packets:

No.	Time	Source	Destination	Protocol	Length	Info
3	1.676423965	127.0.0.1	127.0.0.1	UDP	172	47738 → 0 Len=128
4	1.676434390	127.0.0.1	127.0.0.1	ICMP	200	Destination unreachable (Port unreachable)

Nous utilisons Write() pour écrire dans la socket. Cela ne fonctionne pas car nous ne pouvons préciser le port du serveur sur lequel le paquet doit arriver.  
⇒ Renvoie par le serveur d'un paquet pour signaler une erreur à la réception

Solution : Utiliser la fonction sendto().

# Envoie d'un paquet en UDP

The screenshot displays a development environment with a C program and a network packet capture. The C program, named `getftp.c`, is shown in the editor. It includes a header file `getftp.h` and defines a `sockaddr_in` structure. The program sets up a server address and port (1069) and sends a packet. The network packet capture, titled "Capturing from any", shows two packets. The first packet is a UDP packet from 127.0.0.1 to 127.0.0.1, port 52 56777 to 1069, with a length of 8 bytes. The second packet is a UDP packet from 17.426594178 to 127.0.0.1, port 52 44939 to 1069, with a length of 8 bytes. The packet details show the data as `ff00000000000000` (8 bytes).

```
37  cnar* read_buf[BUF_SIZE];
38  int* RRQ;
39  RRQ = (int*) malloc(BUF_SIZE);
40
41  struct sockaddr_in server;
42  memset(&server, 0, sizeof(server));
43  server.sin_port = htons(1069);
44
45  memset(&read_buf, 0, sizeof(read_buf));
46
47  //Sendto
48
49  *RRQ = (int) 0xFF;
50  //RRQ++
```

2 17.426594178 127.0.0.1 127.0.0.1 UDP 52 44939 → 1069 Len=8  
[Checksum: Unverified]  
[Stream index: 1]  
Data (8 bytes)  
Data: ff00000000000000  
[Length: 8]

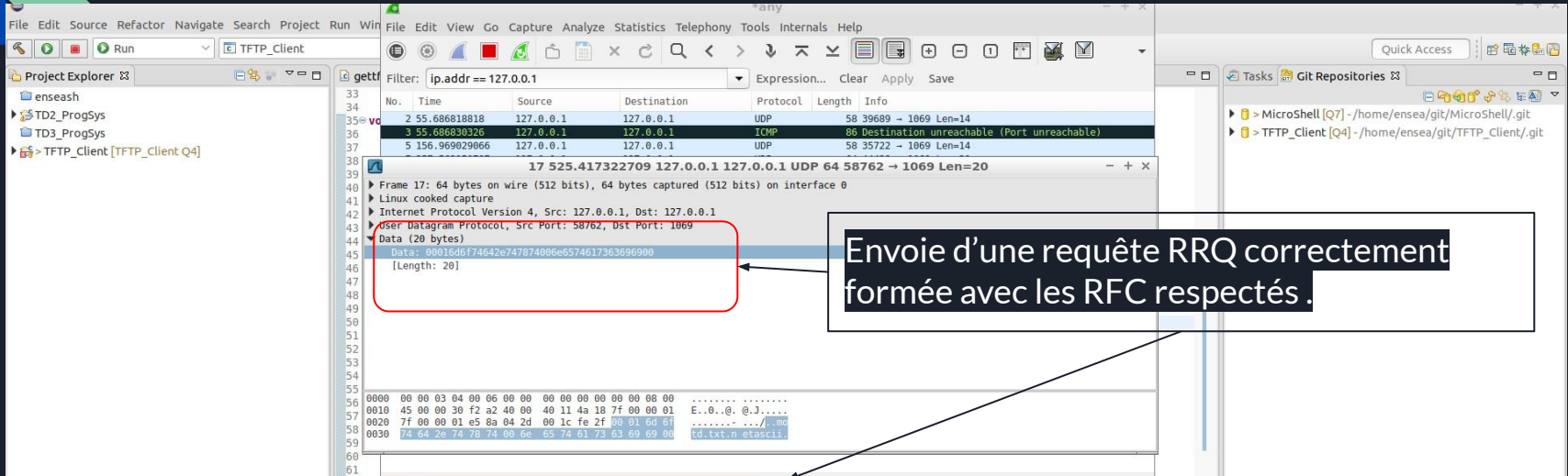
Filter: ip.addr == 127.0.0.1

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	52	56777 → 1069 Len=8
2	17.426594178	127.0.0.1	127.0.0.1	UDP	52	44939 → 1069 Len=8

Nous envoyons un paquet dont le contenu (ff000...0) est défini dans notre code.

Le serveur ne renvoie pas de paquet pour signaler un problème à la réception.  
⇒ Envoie du paquet sur le bon port du serveur

# Envoie d'une requête RRQ



The image shows a Wireshark network capture of an RRQ (Read Request) packet. The packet list on the left shows a UDP packet from 127.0.0.1 to 127.0.0.1 on port 58762. The packet details pane on the right shows the packet structure: Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Data (20 bytes). The data field is highlighted with a red box, and a callout points to it with the text: "Envoie d'une requête RRQ correctement formée avec les RFC respectés."

No.	Time	Source	Destination	Protocol	Length	Info
2	55.686818818	127.0.0.1	127.0.0.1	UDP	58	39689 → 1069 Len=14
3	55.686838326	127.0.0.1	127.0.0.1	ICMP	86	Destination unreachable (Port unreachable)
5	156.969029066	127.0.0.1	127.0.0.1	UDP	58	35722 → 1069 Len=14

Filter: ip.addr == 127.0.0.1

Frame 17: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0

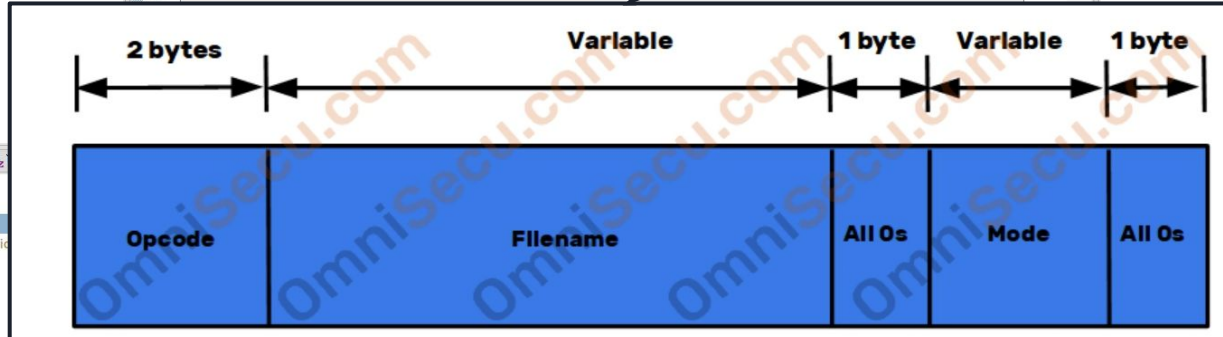
Linux cooked capture

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

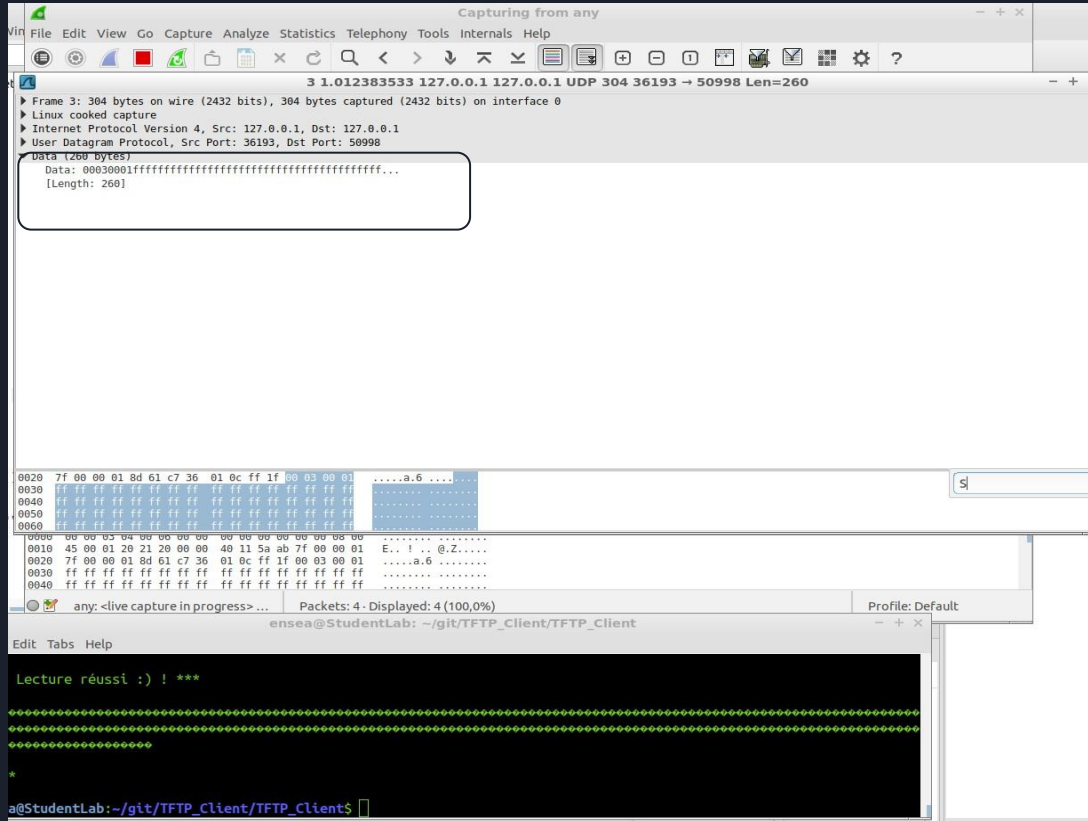
User Datagram Protocol, Src Port: 58762, Dst Port: 1069

Data (20 bytes)

0000 00 00 03 04 00 06 00 00 00 00 00 00 00 00 00 .....  
0010 45 00 00 30 f2 a2 04 00 48 11 4a 18 7f 00 00 01 E..@.@J...  
0020 7f 00 00 01 e5 8a 04 2d 00 1c fe 2f 00 01 6d 61 .....  
0030 74 64 2e 74 78 74 00 6e 65 74 61 73 63 69 69 6e td.txt.n etascii



# Lecture de la réponse du serveur



On reçoit une réponse du serveur contenant le contenu du fichier ones256 avec l'acquittement.

Cela correspond aux premiers octets 01 signalant l'acquittement. Puis suivi par l'ensemble des 256 1.