Théophile Wemaere
Augustin Meaudre Desgouttes

Rapport de TP : Réseaux

Création d'un programme client

Au cours de ce TP, nous avons appris à communiquer avec un serveur à l'aide d'un programme client en C. Nous avons aussi utilisé l'outils nmap pour scanner une machine et trouver ses ports ouverts.

Utilisation de nmap :

L'outils nmap est un outils utilisé pour scanner une machine, détecter ses ports ouverts (en attentes d'une connexion), identifier les services hébergés et même obtenir des informations sur la machine, par exemple le système d'exploitation. Très utilisé dans le domaine de la cybersécurité (pour attaquer une machine, on appelle cela la « nmap enumeration »), l'outils nmap a acquis une certaines réputation, pas toujours très appréciée.

Pour commencer, nous pouvons essayer un scan tout simple sur notre machine (donc sur l'ip 127.0.0.1 = localhost) ; avec la commande *nmap 127.0.0.1* :

```
geii@E211:~$ nmap 127.0.0.1 && netstat -tln
Starting Nmap 7.80 ( https://nmap.org ) at 2021-09-20 15:51 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000079s latency).
Not shown: 995 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
80/tcp  open  http
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds
Connexions Internet actives (seulement serveurs)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN
tcp6       0      0 :::139                  :::*                    LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::445                  :::*                    LISTEN
geii@E211:~$
```

Si l'on compare les données affichées avec celles de la commande *netstat -tln* (vu au précedent TP, permet de voir les services de notre machines en attente de connexion), on remarque que l'on obtient les memes informations, à savoir les memes ports ouverts (excepté pour le protocole TCP6).

L'outils étant très simple à prendre en main, on peut passer à la phase suivante, qui consiste à se connecter au serveur tournant sur la machine à l'adresse 192.168.0.25. Comme lors du TP2, on utilise la commande *telnet ip_adress port* pour se connecter à l'aide du client telnet. Cependant, on ne connait pas le port de la machine sur lequel tourne le serveur. On va donc utiliser nmap.

Dans l'énoncé du TP, on nous indique que le service tourne sur un port compris entre 3000 et 4000. Pour gagner du temps, nmap ne scan qu'une liste de port les plus utilisé pour chaque protocole (445, 80, 22, …). On doit donc lui spécifier de scanner tous les ports, ou un intervalle précis. On peut le faire avec la commande *nmap 192.168.0.25 -p 3000-4000* :

```
geii@E211:~$ sudo nmap 192.168.0.25 -p 3000-4000
Starting Nmap 7.80 ( https://nmap.org ) at 2021-09-20 15:56 CEST
Nmap scan report for 192.168.0.25
Host is up (0.00021s latency).
Not shown: 1000 closed ports
PORT     STATE SERVICE
3456/tcp open  vat
MAC Address: D8:50:E6:49:4F:10 (Asustek Computer)
```

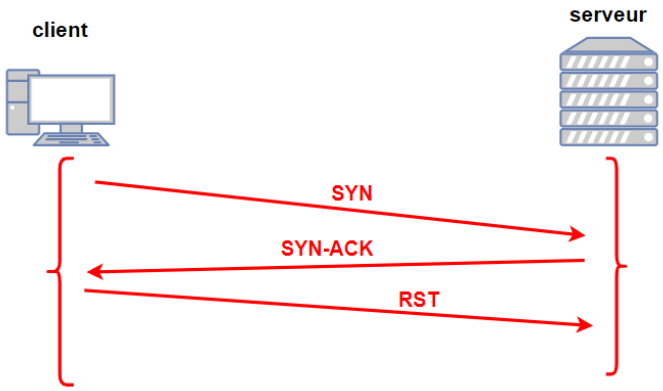
On peut voir que le port 3456 est ouvert, c'est surement celui sur lequel tourne notre serveur. Si l'on n'avait pas spécifié les ports, nmap ne nous afficherait pas le port 3456.

Telnet :

Si l'on tente de se connecter au serveur avec un port fermé, on peut observer sur wireshark les trames suivantes :

192.168.0.33	192.168.0.25	TCP	74 43160 → 3000 [SYN] Seq=0 Win=
192.168.0.25	192.168.0.33	TCP	60 3000 → 43160 [RST, ACK] Seq=1

La première trame (que l'on émet) envois une demande de connexion avec le flag [SYN], mais comme aucun service ne tourne sur le port en question (port 3000), le serveur répond avec le flag [RST, ACK], qui corresponds à une fin brutal de connexion (et une confirmation de la réception de la trame précédente). On peut aussi retrouver le flag [RST] lorsque que l'on coupe une connexion brutalement dans un terminal avec le raccourci ctrl + C. On le retrouve aussi dans certains cas quand un client interroge un serveur puis mets fin brutalement à la connexion pour gagner du temps. On assisterait donc à l'échange de trame suivant :

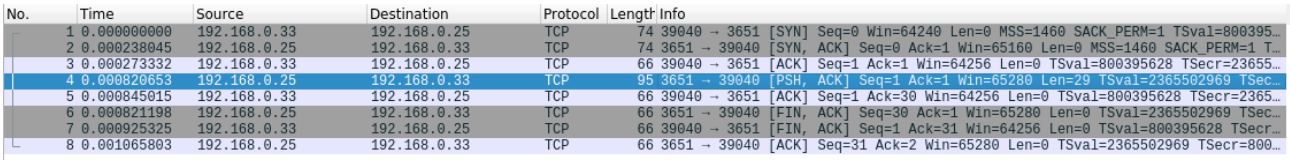


On tente maintenant de se connecter au serveur sur le port 3651 :

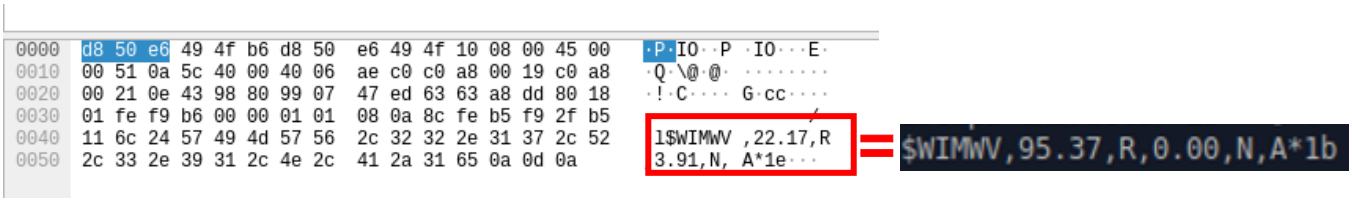
```
geii@E211:~$ telnet 192.168.0.25 3651
Trying 192.168.0.25...
Connected to 192.168.0.25.
Escape character is '^]'.
$WIMWV,95.37,R,0.00,N,A*1b

Connection closed by foreign host.
```

Après la connexion, le serveur nous transmet une trame de données, puis mets fin à la connexion.

Si l'on observe cet échange sur wireshark, on peut observer l'échange de flags habituel :



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.0.33	192.168.0.25	TCP	74	39040 → 3651 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=800395…
2	0.000238045	192.168.0.25	192.168.0.33	TCP	74	3651 → 39040 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 T…
3	0.000273332	192.168.0.33	192.168.0.25	TCP	66	39040 → 3651 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=800395628 TSecr=23655…
4	0.000820653	192.168.0.25	192.168.0.33	TCP	95	3651 → 39040 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=29 TSval=2365502969 TSec…
5	0.000845015	192.168.0.33	192.168.0.25	TCP	66	39040 → 3651 [ACK] Seq=1 Ack=30 Win=64256 Len=0 TSval=800395628 TSecr=2365…
6	0.000821198	192.168.0.25	192.168.0.33	TCP	66	3651 → 39040 [FIN, ACK] Seq=30 Ack=1 Win=65280 Len=0 TSval=2365502969 TSec…
7	0.000925325	192.168.0.33	192.168.0.25	TCP	66	39040 → 3651 [FIN, ACK] Seq=1 Ack=31 Win=64256 Len=0 TSval=800395628 TSecr…
8	0.001065803	192.168.0.25	192.168.0.33	TCP	66	3651 → 39040 [ACK] Seq=31 Ack=2 Win=65280 Len=0 TSval=2365502969 TSecr=800…



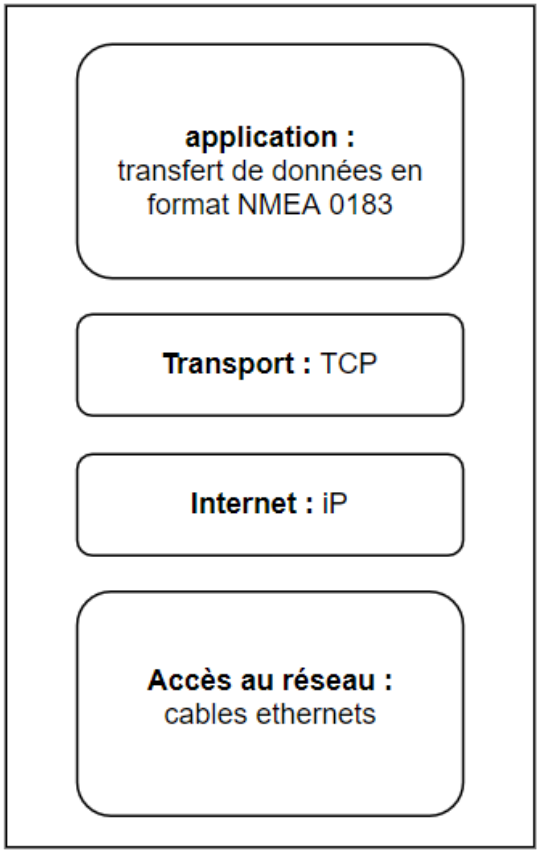
```
0000  d8 50 e6 49 4f b6 d8 50  e6 49 4f 10 08 00 45 00   ·P·IO··P ·IO···E·
0010  00 51 0a 5c 40 00 40 06  ae c0 c0 a8 00 19 c0 a8   ·Q·\@·@· ········
0020  00 21 0e 43 98 80 99 07  47 ed 63 63 a8 dd 80 18   ·!·C···· G·cc····
0030  01 fe f9 b6 00 00 01 01  08 0a 8c fe b5 f9 2f b5   ···· ···· ····/·
0040  11 6c 24 57 49 4d 57 56  2c 32 32 2e 31 37 2c 52   ·l$WIMWV ,22.17,R
0050  2c 33 2e 39 31 2c 4e 2c  41 2a 31 65 0a 0d 0a      3.91,N, A*1e···
```

$WIMWV,95.37,R,0.00,N,A*1b

La connexion s'effectue bien, on retrouve dans la trame avec le flag [PSH] (trame contenant des données) le même format de données que dans le terminal.

La phrase transmises par le serveur est au format NMEA0183 (norme pour la communication entre équipements marins , dont les équipements GPS), de la forme «$--MWV,x.x,a,x.x,a,A*hh» et nous transmets les informations d'une girouette anémomètre (pour récupérer la force et la direction du vent).

On peut donc en déduire l'empilement protocolaire de ce serveur, de modèle TCP/IP:



Programmation d'un client :

Maintenant que nous arrivons à nous connecter au serveur et a récupéré les données de l'anémomètre, il pourrait être pratique de créer un programme qui décode la trame reçu pour nous, afin qu'elle ne nous affiche que les données qui nous intéressent. On va pour cela créer un programme en langage C, qui va devoir :

_créer un socket (interface logicielle qui permet d'exploiter les services d'un protocole réseau)

_se connecter au serveur

_récupérer la trame de données

_fermer le socket

_décoder la trame et l'afficher

_tout recommencer

Nous utiliserons la librairie lib64, qui contient toutes les fonctions nous permettant de nous connecter à un serveur.

Pour commencer, on ajoute à notre projet toutes les libraires nécessaires, donc la lib64 (ici lib.h) et la librairie <unistd.h> qui nous permet d'appeler la fonction *sleep()* pour temporiser notre code (pour éviter de DOS le serveur, de le saturer) :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "lib.h"

char* ip = "192.168.0.25";
int port=3651;
char data[25];
float moy[20];
int a=0;

void clearScreen(void);
void Moy(float* val);


int main(void)
{
      while(1)
      {
```

On en profite aussi pour déclarer variables et tableaux qui nous serviront par la suite, ainsi que le port et l'adresse IP du serveur. Enfin, on déclare les fonctions additionnelles, ici *clearScreen()* pour vider le terminal (pour un affichage des données plus propres) et *Moy()* pour calculer la vitesse moyenne du vent au cours du temps.

Ensuite, on initie notre main() et dans une boucle *while()* infinie (pour récupérer les données en continues), on va pouvoir commencer le processus de connexion. A chaque interaction avec le

serveur, on va effectuer un test pour détecter si une erreur s'est produite. Pour cela, toutes les fonctions de la librairie lib64 retourne -1 comme code d'erreur, ce qui nous permet de le savoir.

Par exemple pour la première fonction *createSocketTCP()* :

```
int s=createSocketTCP();
printf("socket: %d",s);
if(s==-1)
{
   perror("error creating socket");
   return -1;
}
```

Si la fonction retourne -1, on écrit « error creating socket » et on termine le programme.

Sinon la variable s contient le numéro du socket créé.

On réalise la même opération avec la fonction connectServer(), receveData() et closeSocket() :

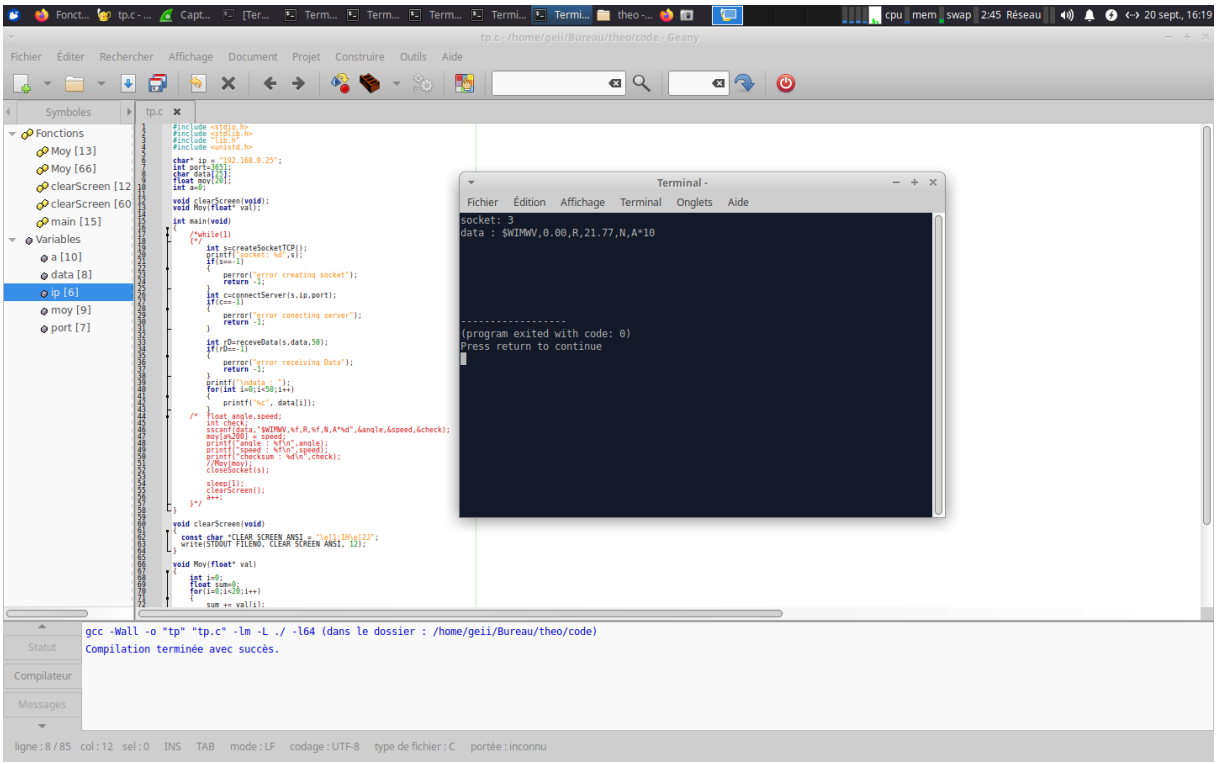
```
int c=connectServer(s,ip,port);
if(c==-1)
{
   perror("error conecting server");
   return -1;
}

int rD=receveData(s,data,25);
if(rD==-1)
{
   perror("error receiving Data");
   return -1;
}

int s2=closeSocket(s);
if(s2==-1)
{
   perror("error closing socket");
   return -1;
}
```

Pour la fonction *connectServer(),* les arguments sont le numéro du socket (s), l'ip du server (ip) et le port sur lequel tourne le serveur (port). Pour la fonction *receveData()*, les arguments sont le numéro du socket (s), le tableau où stocker les données reçues (data) et la taille de la trame (25). Et la fonction *closeSocket()* prend comme argument le numéro de la socket à fermer.

Ensuite, avec les données récupérées, on peut les décoder :

```
printf("\ndata : ");
for(int i=0;i<25;i++)
{
   printf("%c", data[i]);
}
float angle,speed;
int check;
```

On commence par afficher la trame avec une boucle for :



Ensuite, avec la fonction sscanf, on peut indiquer quelle parties de la trame on veut stocker dans des variables.

```
sscanf(data,"$WIMWV,%f,R,%f,N,A*%d",&angle,&speed,&check);
moy[a%200] = speed;
printf("angle : %f\n",angle);
printf("speed : %f\n",speed);
printf("checksum : %d\n",check);
Moy(moy);
sleep(3);
clearScreen();
a++;
```

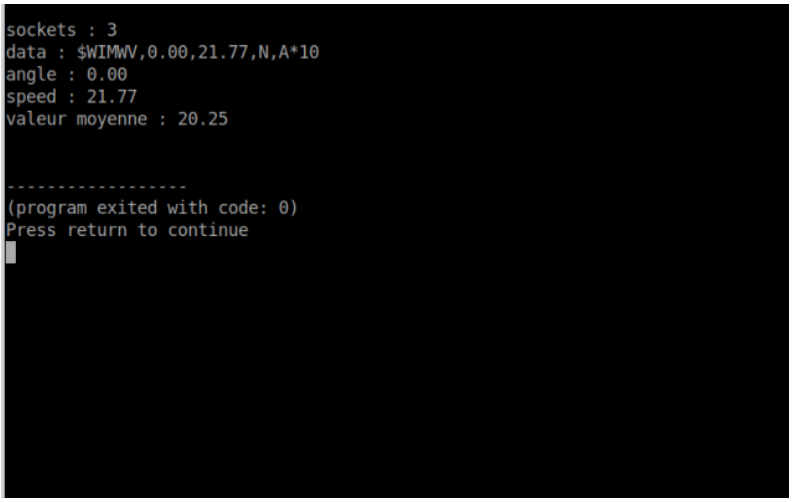
On affiche ensuite toutes les valeurs qui nous intéressent. On remplit également le tableau moy[] pour calculer la moyenne glissante de la vitesse du vent. Ici, on calcul la moyenne sur 20 connexions avec la fonction *Moy()* :

```
void Moy(float* val)
{
    int i=0;
    float sum=0;
    for(i=0;i<20;i++)
    {
        sum += val[i];
    }
    printf("valeur moyenne : %f\n",sum/20);
}
```

Et la fonction *clearScreen()* permet de vider le terminal pour afficher de nouvelles données :

```c
void clearScreen(void)
{
  const char *CLEAR_SCREEN_ANSI = "\e[1;1H\e[2J";
  write(STDOUT_FILENO, CLEAR_SCREEN_ANSI, 12);
}
```

Ainsi, après l'affichage des données, on attend 3 secondes, on nettoie le terminal et on relance l'opération de connexion :



Le code complet : https://github.com/IUT-Theophile-Wemaere/TP-reseau/blob/main/TCP.c

Après avoir laissé le programme tourné pendant quelques minutes, on peut observer l'échange de trames sur wireshark :

No.	Time	Source	Destination	Protocol	Length	Info
255	10.103661330	192.168.0.33	192.168.0.25	TCP	66	45114 → 3651 [FIN, ACK] Seq=1 Ack=39 Win=64256 Len=0 TSval=808478297 TSecr…
256	10.103815919	192.168.0.25	192.168.0.33	TCP	66	3651 → 45114 [ACK] Seq=39 Ack=2 Win=65280 Len=0 TSval=2373585716 TSecr=808…
257	10.188592364	192.168.0.33	192.168.0.25	TCP	74	45116 → 3651 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=808478…
258	10.188819417	192.168.0.25	192.168.0.33	TCP	74	3651 → 45116 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 T…
259	10.188858864	192.168.0.33	192.168.0.25	TCP	66	45116 → 3651 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=808478382 TSecr=23735…
260	10.189384037	192.168.0.25	192.168.0.33	TCP	105	3651 → 45116 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=39 TSval=2373585801 TSec…
261	10.189409024	192.168.0.33	192.168.0.25	TCP	66	45116 → 3651 [ACK] Seq=1 Ack=40 Win=64256 Len=0 TSval=808478382 TSecr=2373…
262	10.189384582	192.168.0.25	192.168.0.33	TCP	66	3651 → 45116 [FIN, ACK] Seq=40 Ack=1 Win=65280 Len=0 TSval=2373585801 TSec…
263	10.189522649	192.168.0.33	192.168.0.25	TCP	66	45116 → 3651 [FIN, ACK] Seq=1 Ack=41 Win=64256 Len=0 TSval=808478383 TSecr…
264	10.189651209	192.168.0.25	192.168.0.33	TCP	66	3651 → 45116 [ACK] Seq=41 Ack=2 Win=65280 Len=0 TSval=2373585801 TSecr=808…
265	11.012160533	192.168.0.33	192.168.0.25	TCP	74	45118 → 3651 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=808479…
266	11.012371861	192.168.0.25	192.168.0.33	TCP	74	3651 → 45118 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 T…
267	11.012406968	192.168.0.33	192.168.0.25	TCP	66	45118 → 3651 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=808479205 TSecr=23735…
268	11.012964353	192.168.0.25	192.168.0.33	TCP	94	3651 → 45118 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=28 TSval=2373586625 TSec…
269	11.012991296	192.168.0.33	192.168.0.25	TCP	66	45118 → 3651 [ACK] Seq=1 Ack=29 Win=64256 Len=0 TSval=808479206 TSecr=2373…
270	11.012964901	192.168.0.25	192.168.0.33	TCP	66	3651 → 45118 [FIN, ACK] Seq=29 Ack=1 Win=65280 Len=0 TSval=2373586625 TSec…
271	11.013101516	192.168.0.33	192.168.0.25	TCP	66	45118 → 3651 [FIN, ACK] Seq=1 Ack=30 Win=64256 Len=0 TSval=808479206 TSecr…
272	11.013218526	192.168.0.25	192.168.0.33	TCP	66	3651 → 45118 [ACK] Seq=30 Ack=2 Win=65280 Len=0 TSval=2373586625 TSecr=808…
273	11.103874946	192.168.0.33	192.168.0.25	TCP	74	45120 → 3651 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=808479…
274	11.104106491	192.168.0.25	192.168.0.33	TCP	74	3651 → 45120 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 T…
275	11.104148281	192.168.0.33	192.168.0.25	TCP	66	45120 → 3651 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=808479297 TSecr=23735…
276	11.104685838	192.168.0.25	192.168.0.33	TCP	96	3651 → 45120 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=30 TSval=2373586716 TSec…
277	11.104713326	192.168.0.33	192.168.0.25	TCP	66	45120 → 3651 [ACK] Seq=1 Ack=31 Win=64256 Len=0 TSval=808479298 TSecr=2373…
278	11.104686381	192.168.0.25	192.168.0.33	TCP	66	3651 → 45120 [FIN, ACK] Seq=31 Ack=1 Win=65280 Len=0 TSval=2373586717 TSec…
279	11.104822546	192.168.0.33	192.168.0.25	TCP	66	45120 → 3651 [FIN, ACK] Seq=1 Ack=32 Win=64256 Len=0 TSval=808479298 TSecr…
280	11.104940036	192.168.0.25	192.168.0.33	TCP	66	3651 → 45120 [ACK] Seq=32 Ack=2 Win=65280 Len=0 TSval=2373586717 TSecr=808…
281	11.189696566	192.168.0.33	192.168.0.25	TCP	74	45122 → 3651 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=808479…
282	11.189929724	192.168.0.25	192.168.0.33	TCP	74	3651 → 45122 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 T…
283	11.189972984	192.168.0.33	192.168.0.25	TCP	66	45122 → 3651 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=808479383 TSecr=23735…
284	11.190502664	192.168.0.25	192.168.0.33	TCP	98	3651 → 45122 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=32 TSval=2373586802 TSec…
285	11.190527701	192.168.0.33	192.168.0.25	TCP	66	45122 → 3651 [ACK] Seq=1 Ack=33 Win=64256 Len=0 TSval=808479384 TSecr=2373…
286	11.190503216	192.168.0.25	192.168.0.33	TCP	66	3651 → 45122 [FIN, ACK] Seq=33 Ack=1 Win=65280 Len=0 TSval=2373586802 TSec…
287	11.190643269	192.168.0.33	192.168.0.25	TCP	66	45122 → 3651 [FIN, ACK] Seq=1 Ack=34 Win=64256 Len=0 TSval=808479384 TSecr…
288	11.190768584	192.168.0.25	192.168.0.33	TCP	66	3651 → 45122 [ACK] Seq=34 Ack=2 Win=65280 Len=0 TSval=2373586803 TSecr=808…

On peut observer tous les processus de connexion et de déconnexion qui se suivent.

La blague de la semaine :