Réseau - StudyMyTopology - 125 points

Kévin DUVERGER

Table des matières

1 Résolution StudyMyTopology :

Résolution à la main. Essayons donc de nous connecter au serveur pour voir ce qu'il nous envoie :

```
Welcome to the network topology challenge !
You will first be given the topology consisting of 3 sections : networks, machines, routes
Each one of the routers applies NAT on packets coming from the network with PCs behind it
Your goal will be to give the path that the packet takes from PC 1 to PC 2
The thing you have to return is a list (srcMACi,srcIPi,dstMACi,dstIPi) for each edge taken on the network (separated by commas)

========== NETWORKS ==========
Network #1 : 192.168.3.0/24
Network #2 : 192.168.113.0/24
Network #3 : 192.168.56.0/24
Network #4 : 192.168.28.0/24
Network #5 : 192.168.205.0/24
Network #6 : 164.133.0.0/16
Network #7 : 164.240.0.0/16
Network #8 : 164.215.0.0/16
Network #9 : 164.84.0.0/16
Network #10 : 164.145.0.0/16

========== MACHINES ==========
Machine PC #45384 :
        Interface #1 : 192.168.3.38, 79:f7:e9:95:9e:a2
Machine PC #13359 :
        Interface #1 : 192.168.3.3, eb:80:100:bc:69:b8
Machine PC #76872 :
        Interface #1 : 192.168.3.45, 85:94:65:7e:cd:ee
Machine PC #16286 :
        Interface #1 : 192.168.3.107, 57:2c:8a:cc:8e:bb
Machine PC #85976 :
        Interface #1 : 192.168.113.35, bb:52:ae:7b:39:48
Machine PC #51088 :
        Interface #1 : 192.168.113.182, fe:ea:82:54:9c:dd
Machine PC #57195 :
        Interface #1 : 192.168.113.204, 10:23:04:12:9a:ba
Machine PC #21816 :
        Interface #1 : 192.168.56.65, e2:f1:f0:17:fb:91
Machine PC #46857 :
        Interface #1 : 192.168.56.6, 08:0b:0c:74:ba:bb
Machine PC #75693 :
        Interface #1 : 192.168.56.109, 0f:b3:85:40:b8:0d
Machine PC #67851 :
        Interface #1 : 192.168.56.44, 31:fe:61:17:e5:49
Machine PC #17966 :
        Interface #1 : 192.168.28.203, 28:cc:ca:6d:08:9b
Machine PC #21832 :
        Interface #1 : 192.168.28.144, 98:f1:09:9b:30:1b
Machine PC #64780 :
        Interface #1 : 192.168.205.181, 04:69:3a:ef:d9:b3
Machine PC #73668 :
        Interface #1 : 192.168.205.174, 19:83:98:a4:c0:a0
Machine Router #70150 :
        Interface #1 : 192.168.3.11, d5:13:e1:66:91:4e
        Interface #2 : 164.133.224.197, 08:98:bd:18:b7:41
        Interface #3 : 164.240.23.249, 30:25:b1:8f:66:3b
Machine Router #46136 :
        Interface #1 : 192.168.113.163, 87:9b:dc:c7:7a:43
        Interface #2 : 164.240.56.2, 74:ba:eb:b2:dc:a1
        Interface #3 : 164.215.150.101, b5:6d:cb:75:ed:95
Machine Router #50956 :
        Interface #1 : 192.168.56.63, 5c:9c:b2:41:71:bd
        Interface #2 : 164.215.190.217, 04:4c:cb:51:2d:f0
        Interface #3 : 164.84.112.72, 79:35:26:d3:25:d3
Machine Router #72503 :
        Interface #1 : 192.168.28.227, 7f:a8:16:e9:c3:7d
        Interface #2 : 164.84.67.191, 9f:11:54:d9:77:57
        Interface #3 : 164.145.118.28, 24:e7:48:0c:35:39
Machine Router #63095 :
        Interface #1 : 192.168.205.199, f7:26:da:d0:8c:38
        Interface #2 : 164.145.22.1, 83:f6:ae:3e:8e:7f
        Interface #3 : 164.133.57.15, 47:04:f2:49:63:14

========== ROUTES ==========
Routes for PC #45384 :
        Network = 192.168.3.0, next hop = 192.168.3.38
        Network = 0.0.0.0, next hop = 192.168.3.11
```

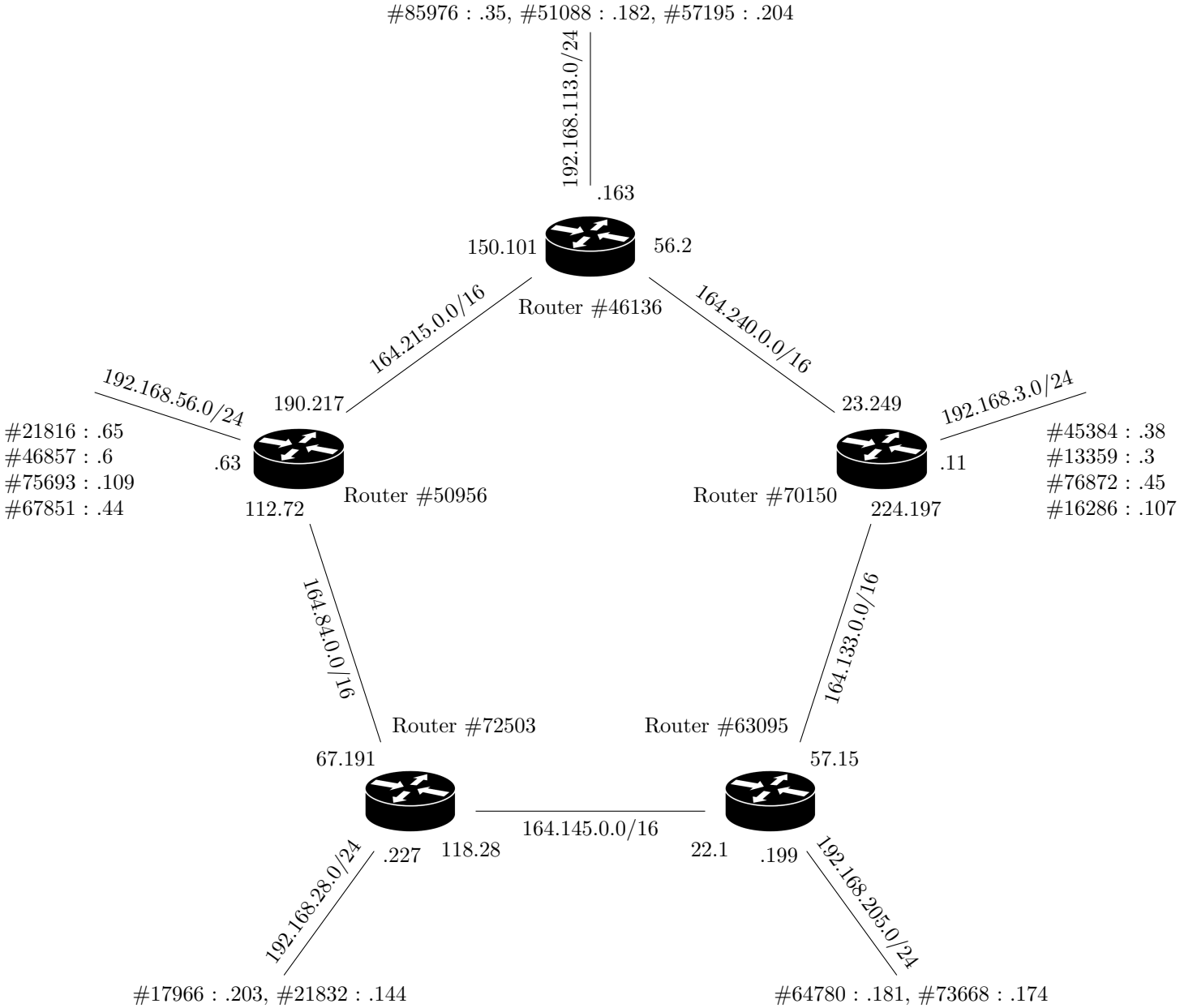
```
Routes for PC #13359 :
        Network = 192.168.3.0, next hop = 192.168.3.3
        Network = 0.0.0.0, next hop = 192.168.3.11
Routes for PC #76872 :
        Network = 192.168.3.0, next hop = 192.168.3.45
        Network = 0.0.0.0, next hop = 192.168.3.11
Routes for PC #16286 :
        Network = 192.168.3.0, next hop = 192.168.3.107
        Network = 0.0.0.0, next hop = 192.168.3.11
Routes for PC #85976 :
        Network = 192.168.113.0, next hop = 192.168.113.35
        Network = 0.0.0.0, next hop = 192.168.113.163
Routes for PC #51088 :
        Network = 192.168.113.0, next hop = 192.168.113.182
        Network = 0.0.0.0, next hop = 192.168.113.163
Routes for PC #57195 :
        Network = 192.168.113.0, next hop = 192.168.113.204
        Network = 0.0.0.0, next hop = 192.168.113.163
Routes for PC #21816 :
        Network = 192.168.56.0, next hop = 192.168.56.65
        Network = 0.0.0.0, next hop = 192.168.56.63
Routes for PC #46857 :
        Network = 192.168.56.0, next hop = 192.168.56.6
        Network = 0.0.0.0, next hop = 192.168.56.63
Routes for PC #75693 :
        Network = 192.168.56.0, next hop = 192.168.56.109
        Network = 0.0.0.0, next hop = 192.168.56.63
Routes for PC #67851 :
        Network = 192.168.56.0, next hop = 192.168.56.44
        Network = 0.0.0.0, next hop = 192.168.56.63
Routes for PC #17966 :
        Network = 192.168.28.0, next hop = 192.168.28.203
        Network = 0.0.0.0, next hop = 192.168.28.227
Routes for PC #21832 :
        Network = 192.168.28.0, next hop = 192.168.28.144
        Network = 0.0.0.0, next hop = 192.168.28.227
Routes for PC #64780 :
        Network = 192.168.205.0, next hop = 192.168.205.181
        Network = 0.0.0.0, next hop = 192.168.205.199
Routes for PC #73668 :
        Network = 192.168.205.0, next hop = 192.168.205.174
        Network = 0.0.0.0, next hop = 192.168.205.199
Routes for Router #70150 :
        Network = 192.168.3.0, next hop = 192.168.3.11
        Network = 164.133.0.0, next hop = 164.133.224.197
        Network = 164.240.0.0, next hop = 164.240.23.249
        Network = 192.168.113.0, next hop = 164.240.56.2
        Network = 164.145.0.0, next hop = 164.133.57.15
        Network = 192.168.205.0, next hop = 164.133.57.15
        Network = 164.215.0.0, next hop = 164.240.56.2
        Network = 192.168.56.0, next hop = 164.240.56.2
        Network = 164.84.0.0, next hop = 164.133.57.15
        Network = 192.168.28.0, next hop = 164.133.57.15
Routes for Router #46136 :
        Network = 192.168.113.0, next hop = 192.168.113.163
        Network = 164.240.0.0, next hop = 164.240.56.2
        Network = 164.215.0.0, next hop = 164.215.150.101
        Network = 192.168.56.0, next hop = 164.215.190.217
        Network = 164.133.0.0, next hop = 164.240.23.249
        Network = 192.168.3.0, next hop = 164.240.23.249
        Network = 164.84.0.0, next hop = 164.215.190.217
        Network = 192.168.28.0, next hop = 164.215.190.217
        Network = 164.145.0.0, next hop = 164.240.23.249
        Network = 192.168.205.0, next hop = 164.240.23.249
Routes for Router #50956 :
        Network = 192.168.56.0, next hop = 192.168.56.63
        Network = 164.215.0.0, next hop = 164.215.190.217
        Network = 164.84.0.0, next hop = 164.84.112.72
        Network = 192.168.28.0, next hop = 164.84.67.191
        Network = 164.240.0.0, next hop = 164.215.150.101
        Network = 192.168.113.0, next hop = 164.215.150.101
        Network = 164.145.0.0, next hop = 164.84.67.191
        Network = 192.168.205.0, next hop = 164.84.67.191
        Network = 164.133.0.0, next hop = 164.215.150.101
        Network = 192.168.3.0, next hop = 164.215.150.101
Routes for Router #72503 :
        Network = 192.168.28.0, next hop = 192.168.28.227
        Network = 164.84.0.0, next hop = 164.84.67.191
        Network = 164.145.0.0, next hop = 164.145.118.28
```

```
        Network = 192.168.205.0, next hop = 164.145.22.1
        Network = 164.215.0.0, next hop = 164.84.112.72
        Network = 192.168.56.0, next hop = 164.84.112.72
        Network = 164.133.0.0, next hop = 164.145.22.1
        Network = 192.168.3.0, next hop = 164.145.22.1
        Network = 164.240.0.0, next hop = 164.84.112.72
        Network = 192.168.113.0, next hop = 164.84.112.72
Routes for Router #63095 :
        Network = 192.168.205.0, next hop = 192.168.205.199
        Network = 164.145.0.0, next hop = 164.145.22.1
        Network = 164.133.0.0, next hop = 164.133.57.15
        Network = 192.168.3.0, next hop = 164.133.224.197
        Network = 164.84.0.0, next hop = 164.145.118.28
        Network = 192.168.28.0, next hop = 164.145.118.28
        Network = 164.240.0.0, next hop = 164.133.224.197
        Network = 192.168.113.0, next hop = 164.133.224.197
        Network = 164.215.0.0, next hop = 164.145.118.28
        Network = 192.168.56.0, next hop = 164.145.118.28

Please give the path between machines PC #76872 and PC #67851 (you have 1 second) : Too late ;)
```

On reçoit donc une assez grosse quantité d'informations qui semble représenter toutes les informations d'une topologie réseau et notre but est de retourner le plus vite possible le chemin entre deux machines.

Essayons donc de la représenter sous forme d'image :



Vous voyez donc que c'est une topologie en forme de polygône, et en relançant le programme plusieurs fois vous devriez voir entre 4 et 8 routeurs (donc du carré à l'octogone). Chaque routeur a son propre réseau avec un petit nombre aléatoire de machines et il est très important de noter qu'il fait de la NAT.

Sur cet exemple, on nous demande de trouver le chemin entre les machines `PC #76872` et `PC #67851` en respectant bien sûr les tables de routage qui nous sont données. Voici la liste des liens qui seront parcourus (en gris, les endpoints) :

Lien	Source			Destination		
-	Machine	MAC	IP	Machine	MAC	IP
1	#76872[1]	85:94:65:7e:cd:ee	192.168.3.45	#70150[1]	d5:13:e1:66:91:4e	192.168.3.11
2	#70150[3]	30:25:b1:8f:66:3b	164.240.23.249	#46136[2]	74:ba:eb:b2:dc:a1	164.240.56.2
3	#46136[3]	b5:6d:cb:75:ed:95	164.215.150.101	#50956[2]	04:4c:cb:51:2d:f0	164.215.190.217
4	#50956[1]	5c:9c:b2:41:71:bd	192.168.56.63	#67851[1]	31:fe:61:17:e5:49	192.168.56.44

La chaine à envoyer au serveur serait donc constituée des quadruplets suivants séparés par une virgule :

- `(85:94:65:7e:cd:ee, 192.168.3.45, d5:13:e1:66:91:4e, 192.168.56.44)`
- `(30:25:b1:8f:66:3b, 164.240.23.249, 74:ba:eb:b2:dc:a1, 192.168.56.44)`
- `(b5:6d:cb:75:ed:95, 164.240.23.249, 04:4c:cb:51:2d:f0, 192.168.56.44)`
- `(5c:9c:b2:41:71:bd, 164.240.23.249, 31:fe:61:17:e5:49, 192.168.56.44)`

Parsing. Pour coder la solution, la première chose qu'il faut est choisir comment représenter chacun des composants :

```
Structure des réseaux :
{
    "IP" : l'adresse IP du réseau sous forme de str,
    "CIDR" : la taille du préfixe réseau
}

Structure des machines :
{
    "ID" : contient l'identifiant de la machine (par exemple "Router #70150"),
    "IFACES" : la liste des interfaces de la machine
}
Structure des interfaces :
{
    "IP" : l'adresse IP de l'interface,
    "MAC" : l'adresse MAC de l'interface
}

Structure de la table des routes d'une machine :
{
    "MACHINE" : l'ID de la machine,
    "ROUTES" : les routes
}
Structure d'une route :
{
    "DEST" : l'adresse IP du réseau de destination,
    "HOP" : l'adresse IP du prochain saut
}
```

Nous pouvons maintenant réaliser la fonction qui va parser les différentes informations depuis l'envoi du serveur :

```
def parseIPFromString(ipstring) :
    temp = ipstring.split(".")
    return [ int(temp[i]) for i in range(0, len(temp)) ]

def parseMACFromString(macstring) :
    temp = macstring.split(":")
    return [ int(temp[i], 16) for i in range(0, len(temp)) ]

def parseTopology(networksStringList, machinesStringList, routesStringList) :
    # First finding the networks
    networks = []
    for i in range(0, len(networksStringList)) :
        temp = networksStringList[i].split(" : ")
        temp = temp[1].split("/")
        networks.append({ "IP" : parseIPFromString(temp[0]), "CIDR" : int(temp[1]) })

    # Then, finding the machines
    machines = []
    currentMachineID, currentMachineIFACES = None, None
    for i in range(0, len(machinesStringList)) :
        if machinesStringList[i][:7] == "Machine" :
            if currentMachineID != None : # We add it to the list
```

```
                machines.append({ "ID" : currentMachineID, "IFACES" : currentMachineIFACES })
            currentMachineID = machinesStringList[i][8:-3]
            currentMachineIFACES = []
        else :
            temp = machinesStringList[i].split(" : ")
            temp = temp[1].split(", ")
            currentMachineIFACES.append({ "IP" : parseIPFromString(temp[0]), "ETHER" :
    parseMACFromString(temp[1]) })
    machines.append({ "ID" : currentMachineID, "IFACES" : currentMachineIFACES })

    # Finally, finding the routes
    routes = []
    currentMachine, currentRoutes = None, None
    for i in range(0, len(routesStringList)) :
        if routesStringList[i][:11] == "Routes for " :
            if currentMachine != None :
                routes.append({ "MACHINE" : currentMachine, "ROUTES" : currentRoutes })
            currentMachine = routesStringList[i][11:-3]
            currentRoutes = []
        else :
            temp = routesStringList[i].split(", ")
            temp[0] = temp[0].split(" = ")[1]
            temp[1] = temp[1].split(" = ")[1]
            currentRoutes.append({ "DEST" : parseIPFromString(temp[0]), "HOP" : parseIPFromString(
    temp[1]) })
    routes.append({ "MACHINE" : currentMachine, "ROUTES" : currentRoutes })

    # Returning the result
    return networks, machines, routes
```

Trouver le chemin. Une fois que l'on a réussi à parser les données envoyées par le serveur, il ne reste plus qu'à trouver le chemin entre les 2 machines. L'idée du code ci-dessous est de commencer par trouver un chemin constitué d'index de machines : pour passer d'une machine à une autre on cherche le réseau de destination dans la table des routes de la machine actuelle et on prend le next hop associé. Une fois cette liste trouvée, on peut chercher le réseau commun entre chaque paire de machines pour trouver les interfaces, ce qui peremt de conclure. Voici donc le code qui permet d'aboutir à cela :

```
def getNetworkIndex(networks, IP) :
    for i in range(0, len(networks)) :
        testCount   = networks[i]["CIDR"] // 8
        prefixEqual = True
        for j in range(0, testCount) :
            if IP[j] != networks[i]["IP"][j] :
                prefixEqual = False
                break
        if prefixEqual :
            return i
    return -1

def getMachineIndex(machines, IP) :
    for i in range(0, len(machines)) :
        for j in range(0, len(machines[i]["IFACES"])) :
            if IP == machines[i]["IFACES"][j]["IP"] :
                return i
    return -1

def findPath(machine1Index, machine2Index, networks, machines, routes) :
    # Finding the path in machine indexes
    path1 = [ machine1Index ]
    while True :
        # First, finding the route to take
        routeIndexToTake = -1
        currMachineRoutes = routes[path1[-1]]["ROUTES"]
        for i in range(0, len(currMachineRoutes)) :
            equal = True
            currComparisonIndex = 0
            while currComparisonIndex < 4 and currMachineRoutes[i]["DEST"][currComparisonIndex] != 0
     :
                if machines[machine2Index]["IFACES"][0]["IP"][currComparisonIndex] !=
    currMachineRoutes[i]["DEST"][currComparisonIndex] :
                    equal = False
                    break
                currComparisonIndex += 1
            if equal :
                routeIndexToTake = i
        if routeIndexToTake == -1 :
            print("WARNING : no route was found !")
        # Now, we can find the machine associated to the hop
```

```
        nextMachineIndex = getMachineIndex(machines, currMachineRoutes[routeIndexToTake]["HOP"])
        if nextMachineIndex == -1 :
            print("WARNING : no next machine was found !")
        if nextMachineIndex != path1[-1] :
            path1.append(nextMachineIndex)
        else :
            path1.append(machine2Index)
            break

    # Converting this path to the quadruplets asked by the question
    firstRouterIP = None
    toReturn = []
    for i in range(0, len(path1) - 1) :
        # First, finding the only corresponding IP index between the two
        corresponding1, corresponding2 = -1, -1
        for j in range(0, len(machines[path1[i]]["IFACES"])) :
            for k in range(0, len(machines[path1[i + 1]]["IFACES"])) :
                if getNetworkIndex(networks, machines[path1[i]]["IFACES"][j]["IP"]) ==
    getNetworkIndex(networks, machines[path1[i + 1]]["IFACES"][k]["IP"]) :
                    corresponding1, corresponding2 = j, k
                    if i == 0 :
                        firstRouterIP = machines[path1[i + 1]]["IFACES"][k]["IP"]
        toReturn.append([
            machines[path1[i]]["IFACES"][corresponding1]["ETHER"],
            machines[machine1Index]["IFACES"][0]["IP"] if i == 0 else firstRouterIP,
            machines[path1[i + 1]]["IFACES"][corresponding2]["ETHER"],
            machines[machine2Index]["IFACES"][0]["IP"]
        ])
    return toReturn
```

Le main. On peut ensuite terminer par le code du programme principal qui va se charger de tout combiner ensemble (parser les données du serveur, puis trouver le chemin et le renvoyer) :

```
def ipToString(IP) :
    return str(IP[0]) + "." + str(IP[1]) + "." + str(IP[2]) + "." + str(IP[3])

def macToString(mac) :
    return ":".join([ "{:02x}".format(mac[i]) for i in range(0, 6) ])

def findMachineIndexFromID(machineID, machines) :
    for i in range(0, len(machines)) :
        if machines[i]["ID"] == machineID :
            return i
    return -1

server = "146.59.227.136"
port = 23412

# First step, connecting to the server
ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

# Getting the first input and parsing it
data = ""
while "second" not in data :
    data += ma_socket.recv(65536).decode("utf-8")
lines = data.split("\n")
indexNetworks = lines.index("========== NETWORKS ==========")
indexMachines = lines.index("========== MACHINES ==========")
indexRoutes   = lines.index("========== ROUTES ==========")
networks, machines, routes = parseTopology(lines[indexNetworks+1:indexMachines - 1], lines[
    indexMachines+1:indexRoutes - 1], lines[indexRoutes+1:-2])
idMachine1 = lines[-1][38:47]
idMachine2 = lines[-1][52:61]
machine1Index, machine2Index = findMachineIndexFromID(idMachine1, machines), findMachineIndexFromID(
    idMachine2, machines)

# Computing the expected path
result = findPath(machine1Index, machine2Index, networks, machines, routes)
resultStr = ""
for i in range(0, len(result)) :
    resultStr += "(" + macToString(result[i][0]) + "," + ipToString(result[i][1]) + "," +
    macToString(result[i][2]) + "," + ipToString(result[i][3]) + ")"
    if i != len(result) - 1 :
        resultStr += ","
```

```
ma_socket.sendall((resultStr + "\n").encode("utf-8"))

print(ma_socket.recv(65536).decode("utf-8"))

# Closing
ma_socket.close()
```