BSM 307 BİLGİSAYAR AĞLARI GÜZ 2021 DÖNEM PROJESİ
RAPORU:

CONTROLLER:

Eğer router sayısı 1 ve 2 olursa manuel olarak oluşturuluyor.

Eğer router sayısı 2’den fazla ise algoritma devreye girip kendi kendine patterini oluşturuyor.

0: Server

n+1: Client

routerSchemaGenerator()

n değişkeni düğüm sayısını söylüyor.

Düğüm sayısana göre belirli bir pattern oluşturuyoruz.

Bu düğümlerin birbirine bağlı olup olmadığınıda hesaplıyoruz.

Düğümler 2-1-2-1 şeklinde ilerleyeceği için bununda büyüklüğünü tutuyoruz ilerde paket iletirken rahatlık sağlasın diye.

'''

'''

routerSchemaGenerator()

Bu fonksiyon benzersiz ulaşım patternini oluşturuyor.

İlk başta routerData objectini tanımlayıp router kada obje oluşturucak.

Patterndeki her bir sütun için width değeri bir artıyor.

'''

Eğer router sayısı 1 ve 2 olursa manuel olarak oluşturuluyor.

Eğer router sayısı 2 den fazla ise algoritma devreye girip kendi kendine patterini oluşturuyor.

0: Server

n+1: Client

İlk başta algoritma router sayısını kalansız olarak 3 e bölüyor.

Bölümün amacı algoritma router düğümlerini 3'er 3'er oluşturuyor.

Geri kalan düğümler ise total router sayısından 3'lü düğüm sayısının farkı diff değişkenine aktarılıyor.

3'lü pattern sayısı birden fazla ise for döngüsü ile otomatik oluşturuluyor.

Sadece bir tane 3lü node var ise manuel oluşturuluyor.

3'lü pattern sayısı kadar for döngüsü dönüyor.

1.Düğüm en üstte kalıyor.

2.Düğüm ortada kalıyor.

3.Düğüm en altta kalıyor.

Örnek 7 düğümlü bir pattern oluşturalım.

2 Kez for döngüsü olacak.

1.For döngüsü

- 1.Düğüm Koşulları, Düğümler:

1 - Şimdi 1-3 = -2 olduğundan, -2.node yok bu yüzden 0 alıyoruz. Sebebi ise 0'ı server olarak görüyoruz altında kalanları server olarak alıyoruz.

Eğer değer - değilde + olarak geliyorsa düğüm + 3 olarak alıyoruz.

2 - Her zaman 3'lü düğümde 1. düğümün 2 fazlası olcağından sabit bir olasılık olarak kabul ediyoruz.

3 - 1.düğümdeki gibi kontrol sağlayıp aynı koşulları sağlıyoruz.

- 2.Düğüm Koşulları, Düğümler:

1 - Şimdi 1-3 = -2 olduğundan, -2.node yok bu yüzden 0 alıyoruz. Sebebi ise 0'ı server olarak görüyoruz altında kalanları server olarak alıyoruz.

Eğer değer - değilde + olarak geliyorsa düğüm + 3 olarak alıyoruz.

2 - Her zaman 3'lü düğümde 1. düğümün 1 fazlası olcağından sabit bir olasılık olarak kabul ediyoruz.

3 - 1.düğümdeki gibi kontrol sağlayıp aynı koşulları sağlıyoruz.

- 3.Düğüm Koşulları, Düğümler:

1 - Düğümün 1 eksiği alıyoruz.

2 - Düğümün 2 eksiğini alıyoruz Burdaki 3 olasılık sabit olasılıktır.

3 - Düğümün 1 fazlasını alıyoruz.

4 - Düğümün 2 fazlası Router Sayısı + 2 sayısından büyük veya eşit olursa boş değer dönüyor.

Koşul sağlanmazsa düğümün 2 fazlası sağlanır.

2.For döngüsü

- 1.Düğüm Koşulları, Düğümler:

1 - Şimdi 1-3 = -2 olduğundan, -2.node yok bu yüzden 0 alıyoruz. Sebebi ise 0'ı server olarak görüyoruz altında kalanları server olarak alıyoruz.

Eğer değer - değilde + olarak geliyorsa düğüm + 3 olarak alıyoruz.

2 - Her zaman 3'lü düğümde 1. düğümün 2 fazlası olcağından sabit bir olasılık olarak kabul ediyoruz.

3 - 1.düğümdeki gibi kontrol sağlayıp aynı koşulları sağlıyoruz.

- 2.Düğüm Koşulları, Düğümler:

1 - Şimdi 1-3 = -2 olduğundan, -2.node yok bu yüzden 0 alıyoruz. Sebebi ise 0'ı server olarak görüyoruz altında kalanları server olarak alıyoruz.

Eğer değer - değilde + olarak geliyorsa düğüm + 3 olarak alıyoruz.

2 - Her zaman 3'lü düğümde 1. düğümün 1 fazlası olcağından sabit bir olasılık olarak kabul ediyoruz.

3 - 1.düğümdeki gibi kontrol sağlayıp aynı koşulları sağlıyoruz.

- 3.Düğüm Koşulları, Düğümler:

1 - Düğümün 1 eksiği alıyoruz.

2 - Düğümün 2 eksiğini alıyoruz Burdaki 3 olasılık sabit olasılıktır.

3 - Düğümün 1 fazlasını alıyoruz.

4 - Düğümün 2 fazlası Router Sayısı + 2 sayısından büyük veya eşit olursa boş değer dönüyor.

Koşul sağlanmazsa düğümün 2 fazlası sağlanır.

6 Tane node oluşturduk. Geriye bir düğüm kaldı. Bu düğümüde manuel oluşturucaz.

Son düğüm ise bir önceki düğüm ve client arasında seri bağlı olacağından şuanki düğüm numarasının 1 eksik ve fazlasına bağlı olur.

```python
def routerSchemaGenerator(n):
    routerData = {
        "nodes": n,
        "width": 0,
        "routerNodes": []
    }
    width = 0
    nodeId = 1

    '''...'''

    if n == 1:
        routerData["routerNodes"].append({"nodeId": 1, "pairedNodes": "0,2"})
        width = width + 1
    elif n == 2:
        routerData["routerNodes"].append({"nodeId": 1, "pairedNodes": "0,2"})
        routerData["routerNodes"].append({"nodeId": 2, "pairedNodes": "1,3"})
        width = width + 2
    else:
        '''...'''
        tempPatrn = n // 3
        diff = n - (tempPatrn * 3)

        '''...'''
        if tempPatrn > 1:...
        else:
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "0,{},{}".format((nodeId + 2), (nodeId + 3 if
            nodeId = nodeId + 1
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "0,{},{}".format((nodeId + 1), (nodeId + 3 if
            nodeId = nodeId + 1
```

```python
        '''...'''
        tempPatrn = n // 3
        diff = n - (tempPatrn * 3)

        '''...'''
        if tempPatrn > 1:...
        else:
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "0,{},{}".format((nodeId + 2), (nodeId + 3 i
            nodeId = nodeId + 1
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "0,{},{}".format((nodeId + 1), (nodeId + 3 i
            nodeId = nodeId + 1
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "{},{},{},{}".format((nodeId - 2), (nodeId
            nodeId = nodeId + 1
            width = width + 2


        if diff == 2:
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "{},{},{}".format((nodeId - 3), (nodeId - 1)
            nodeId = nodeId + 1
            width = width + 1
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "{},{},{}".format((nodeId - 3), (nodeId - 2)
            nodeId = nodeId + 1
            width = width + 1
        elif diff == 1:
            routerData["routerNodes"].append({"nodeId": nodeId, "pairedNodes": "{},{}".format((nodeId - 1),(nodeId + 1))})
            nodeId = nodeId + 1
            width = width + 1

    routerData["width"] = width

```

Burdaki fonksiyon yukarda oluşturduğumuz değişkeni alarak routerlerin uygunluk durumuna göre uygun rota belirliyoruz.

Her düğüm bağlı olduğu düğümlerin listesi alır. Bu listede tüm düğümlerin buffer kapasiteleri kontrol edilir.

Bu kontrol sonucu uygun olan düğümler başka listeye atanır.

Bu düğümler arasından rastgele birisi seçilerek yolun belirlendiği listeye atanır.

Bir sonraki kontrolde ise en son düğüm alınıp yol listesinden yukardaki işlemler tekralanır.

Bu işlem tekrarlanması routerSayısı+1 yani cliente kadar devam eder.

Cliente ulaşıldıktan sonra rota yani yol döndürülür.

router sayısı 1 veya 2 ise manuel olarak kontrol ediyoruz.

Her routerin buffer kapasitesini kontrol edip ona göre yol belirlemesi yapıyor.

Eğer router sayısı 2'den büyükse burda ilk 2 router kontrol edilip uygunluk durumunu göre ikisinden birisi seçilir.

Sütun sayısına göre yol belirliyoruz. 4 sütundan oluşuyorsa 4 aşamalı yol olacak.

Şuanki düğümü router yolumuzdaki son düğüm olarak alıyoruz.

Düğümün tüm verilerini çekmek için routerSchemadaki indisine göre alıyoruz

Örnek 3.düğümde isek arraylerden 0'dan başladığı için 2.düğümü alıyoruz.

Sonra aldığımız veriden bağlı olduğu düğüm bilgisini alıyoruz.

Aldığımız düğüm verisi ',' karakteri ile bölüp arraye dönüştürüyoruz.

Eğer for döngüsünde başlangıç değerimiz router değerimizin bir fazlasına eşitse client olarak ekliyoruz. Elif de ise paket ileri yönlü olarak gideceği için kendi düğüm numarasından büyük düğüm numaralarını listeye atıyoruz.

Döngüde olası düğüm listesindeki düğüm numaraları ile buffer kapasitelerini kontrol ediyoruz. Buffer değeri dolu olanları listeden çıkarıyoruz.

Olası düğüm listesindeki düğümler arasında rastgele seçim yapılarak yönlendiricilerin kullanacağı yol belirlenir.

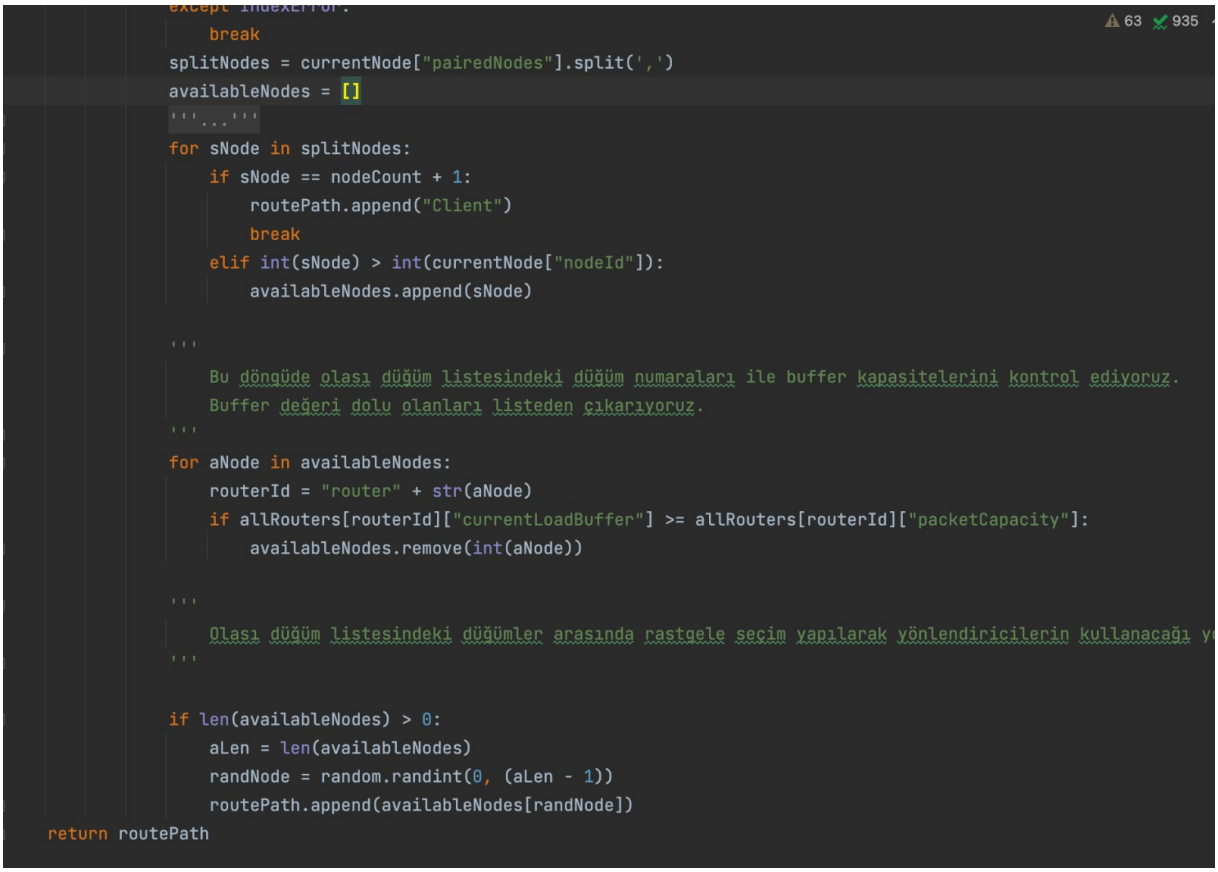
```python
def createRouterPath(routerSchema):
    nodeCount = len(routerSchema["routerNodes"])
    routerPath = {
        "message": "",
        "route": ""
    }

    routePath = []
    '''...'''
    if nodeCount == 1:
        if allRouters["router1"]["currentLoadBuffer"] >= allRouters["router1"]["packetCapacity"]:
            print("Router1 kapasitesi dolu")
        else:
            routePath.append(1)
            routePath.append(2)
    elif nodeCount == 2:
        if allRouters["router1"]["currentLoadBuffer"] >= allRouters["router1"]["packetCapacity"]:
            print("Router1 kapasitesi dolu")
        else:
            if allRouters["router2"]["currentLoadBuffer"] >= allRouters["router2"]["packetCapacity"]:
                print("Router2 kapasitesi dolu!")
            else:
                routePath.append(1)
                routePath.append(2)
                routePath.append(3)
    else:
        '''...'''
        if not allRouters["router1"]["currentLoadBuffer"] <= allRouters["router1"]["packetCapacity"]:
            print("Router1 dolu")
        elif not allRouters["router2"]["currentLoadBuffer"] <= allRouters["router2"]["packetCapacity"]:
```

```python
        '''...'''
        if not allRouters["router1"]["currentLoadBuffer"] <= allRouters["router1"]["packetCapacity"]:
            print("Router1 dolu")
        elif not allRouters["router2"]["currentLoadBuffer"] <= allRouters["router2"]["packetCapacity"]:
            print("Router2 dolu")
        else:
            routePath.append(random.randint(1,2))

        '''
            Burdaki sütun sayısına göre yol belirliyoruz. 4 sütundan oluşuyorsa 4 aşamalı yol olacak.
        '''
        for _ in range(0, int(routerSchema["width"])):
            '''...'''
            if len(routePath) > 0:
                '''...'''
                currentPathNode = routePath[-1]

                try:
                    currentNode = routerSchema["routerNodes"][int(currentPathNode) - 1]
                except IndexError:
                    break
                splitNodes = currentNode["pairedNodes"].split(',')
                availableNodes = []
                '''...'''
                for sNode in splitNodes:
                    if sNode == nodeCount + 1:
                        routePath.append("Client")
                        break
                    elif int(sNode) > int(currentNode["nodeId"]):
                        availableNodes.append(sNode)
```


63
935
except IndexError:
break
splitNodes = currentNode["pairedNodes"].split(',')
availableNodes = []
'''...'''
for sNode in splitNodes:
if sNode == nodeCount + 1:
routePath.append("Client")
break
elif int(sNode) > int(currentNode["nodeId"]):
availableNodes.append(sNode)
'''
Bu döngüde olası düğüm listesindeki düğüm numaraları ile buffer kapasitelerini kontrol ediyoruz.
Buffer değeri dolu olanları listeden çıkarıyoruz.
'''
for aNode in availableNodes:
routerId = "router" + str(aNode)
if allRouters[routerId]["currentLoadBuffer"] >= allRouters[routerId]["packetCapacity"]:
availableNodes.remove(int(aNode))
'''
Olası düğüm listesindeki düğümler arasında rastgele seçim yapılarak yönlendiricilerin kullanacağı y
'''
if len(availableNodes) > 0:
aLen = len(availableNodes)
randNode = random.randint(0, (aLen - 1))
routePath.append(availableNodes[randNode])
return routePath

SERVER:

checkOnlineClients()

Burdaki fonksiyonumuz yukardaki client listesini kullanarak online durumunu denetliyor.

Basitçe for döngüsüne sokup online değiller ise socket oluşturuyoruz.

Bu geçici soket sayesinde bağlanıp ve bağlantının başarılı olduğunu anlayınca online durumunu değiştiriyoruz.

Sonra soketimizi kapatıp ekrana "client1 aktif" şeklinde yazı bastırıyoruz.

```
onlineClients = []

def checkOnlineClients():
    for key in allClients.keys():
        if not allClients[key]["clientOnline"]:

            socketSend = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            try:
                socketSend.connect(('localhost', allClients[key]["clientPort"]))
                allClients[key]["clientOnline"] = True
                print(key, "aktif.")
            except socket.error as exc:
                print(key, "pasif.")

            socketSend.close()
        else:
            print(key, "pasif.")
'
```

Sonsuz döngüye sokarak burada gelen bağlantıdaki portu denetliyoruz.

Gelen bağlantıdaki port kontrolcü portumuzla eşleşiyorsa işlemlere başlıyoruz.

online kullanıcı sayımız bir veya birden fazla ise işlemlere başlıyoruz.

Normalde arrayler 0'dan başladığı için Kullanıcı0 yazdırmak mantıksız olacağı için

indis değerimizin bir fazlası olarak Kullanıcı1 yazdırıyoruz.

[0] => Kullanıcı1

[1] => Kullanıcı2

[2] => Kullanıcı3

Kullanıcıdan hangi kullanıcıya mesaj göndermesi isteyeceğini seçtiriyoruz. try except ile gelen verini int olup olmadığımı kontrol ediyoruz.

Kullanıcımızdan bir mesaj istiyoruz. Bu mesaj boş bir değerse None ile değiştiriyoruz. Bunun amacı ise ilerde paketi parçalarken bir sorun çıkmasını önlemek için.

paketimizi oluşturmaya başlıyoruz.

IPHeader kısmında Gönderici-Alıcı mantığı ile oluşturuyoruz.

Örnek: 192.168.1.1-192.168.1.20:4000

MacHeader kısmında Gönderici-Alıcı mantığı ile oluşturuyoruz.

Örnek: DF:ER:3D:4T:QW-WE:R3:T5 Şeklinde

Paketimiz artık: MacHeader|IPHeader|Paket

şeklinde oluştu.

Paketimizi gelen bağlantıya .send() metodu ile gönderiyoruz.

Burda oluşturuduğumuz paketi byte tipinde gönderiyoruz ve UTF-8 formatında gönderiyoruz.

```
while True:
    if address[1] == 1198:
        if len(onlineClients) > 0:
            print("-Kullanıcılar-")
            for clientIndex, client in enumerate(onlineClients):
                print("{}-) {}".format((clientIndex+1), "Kullanıcı" + str(clientIndex+1)))

            try:
                selectedClient = int(input("Mesaj gönderceğiniz kullanıcı seçin: "))
            except ValueError:
                print("Lütfen Geçerli bir değer girin!")
                break

            getMessage = input("Mesajınızı girin: ")

            if getMessage == '':
                getMessage = "None"
            try:
                ipHeader = serverIp + "-" + onlineClients[selectedClient - 1]["clientAddress"] + ":" + str(
                onlineClients[selectedClient - 1]["clientPort"])
                ethernetHeader = serverMac + "-" + onlineClients[selectedClient - 1]["clientMacAddress"]
                packet = ethernetHeader + "|" + ipHeader + "|" + getMessage
                connection.send(bytes(packet, "UTF-8"))
            except:
                print("Geçerli client değeri girin!")

        else:
            print("Tüm kullanıcılar pasif!")
            print("Sunucu kapatılıyor!")
            server.close()
            connection.close()
            sys.exit(-1)
```

ROUTER:

Burada router bilgilerimizi tutuyoruz.

Router bilgilerimizi tutma sebebimiz gelen paketti veriler ile kendi router bilgilerimizi eşleştirip daha rahat işlem yapabilmek.

Bilgilerimizde paket kapasitemiz, şu an kaç paket olduğu, online durumu ve router IP, Mac gibi bilgileri barındırıyoruz.

```
routerDetails = {
    "routerUUID": "14b5393b-a107-4144-85b7-465a20e9a390",
    "packetCapacity": 100,
    "currentLoadBuffer": 0,
    "packetMissRate": 5,
    "isOnline": False,
    "routerAddress": "10.0.1.1",
    "routerReceivePort": 1200,
    "routerSendPort": 1201,
    "routerMacAddress": "0c:f6:fd:10:0f:42"
}
```

RIH veya RMH değerlerini alırken [4:] ifadesini kullandık.

Bunun sebebi ise şu şekilde;

Şimdi biz RIH:Ip Adresi şeklinde header kullanıyoruz.

Ip adreslerini ayrıştırırken "RIH:" ifadesi engel teşkil ediyor o yüzden ilk 4 karakteri almıyoruz.

```
            routeIpHeader = splitRouteHeader[1][4:]
            routeMacHeader = splitRouteHeader[0][4:]
            splitIpHeader = routeIpHeader.split(',')
            splitMacHeader = routeMacHeader.split(',')
```

Header bilgilerini ve mesajı kullanarak bir paket oluşturuyoruz.

Örnek paket |kontrolcüMac-routerMac|kontrolcüIprouterIp|RMH|RIH|Mesaj

```
fullHeader = "|" + routerDetails["routerMacAddress"] + "-" + splitMacHeader[selectedRouterIndex] + "|"
fullHeader = fullHeader + routerDetails["routerAddress"] + "-" + selectedIp + "|"
fullHeader = fullHeader + splitRouteHeader[0] + "-" + splitRouteHeader[1] + "|"
messagePacket = fullHeader + splitMessage[-1]
```

Burada geçici bir soket tanımlayıp bunu ilk noktamızın portuna bağlıyoruz.

Bağlantı gerçekleştikten sonra paketi yolluyoruz.

Paket yollandıktan sonra soketimizi kapatıyoruz.

```
    routerSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        routerSocket.connect(('localhost', int(selectedPort)))
        routerSocket.send(bytes(messagePacket, "UTF-8"))
        routerSocket.shutdown(0)
    except socket.error:
        print(socket.error)
```

```
        print("----------------")
        print("Mesaj geldi! Detaylar:")
        print("Gönderen IP:", routerDetails["routerAddress"])
        print("Alıcı IP:", selectedIp)
        print("Gönderen MAC:", routerDetails["routerMacAddress"])
        print("Alıcı MAC:", splitMacHeader[selectedRouterIndex])
        print("Mesaj:", splitMessage[-1])
        print("----------------")

        connection.close()
else:
    print("Bağlantı gelmedi")
```

.bind() metodu ile sunucumuza ip adresini ve portunu veriyoruz. Bu sayede sunucumuz kendi ip adresini ve portunu biliyor. .listen() ile gelen bağlantıları dinliyoruz. Gelen bağlantılara göre işlem yapmamız gerekecek.

```
router = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
router.bind(('localhost', routerDetails["routerReceivePort"]))

router.listen(4)
```

CLİENT:

Burda sınırsız döngüde gelen bağlantıları .accept() metodu ile kabul ediyoruz.

.accept() metodu bize bir array döndürüyor.

[0] => Bağlantı bilgileri

[1] => Bağlantı yapanın adres bilgileri

[1][0] => Ip Adresi

[1][1] => Port

Bu bilgileri kullanarak işlemlerimize devam ediyoruz.

Bağlantının geçerli olup olmadığını kontrol ediyoruz. Eğer bağlantımız geçerli ise gelen mesajı denetliyoruz. Gelen mesajımız boş bir mesaj değil ise işlemlerimize devam ediyoruz.

Gelen mesajımızı UTF-8 formatında decode ediyoruz.

Şimdi gelen mesaj aslında paketimiz.

Paketimizin şeması şu şekilde idi: MacHeader|IPHeader|RMH|RIH|Paket

RIH = Route Ip Header

RMH = Route Mac Header

Bu paketteki bilgileri alıp daha rahat işlem yapmak için split metodu ile '|' karakterini bölüyoruz.

Bazen bazı paketlerde sorun çıkabiliyor. Boş string elemanları olabiliyor.

Bu sorunu kaldırmak içinde boş elemanları kaldırıyoruz.

Split ettiğimizde elimizde bir array var.

[0] => Mac Header

[1] => IP Header

[2] => RIH Route IP Header

[3] => RMH Route Mac Header

[4] => Mesaj

Bu bilgileri kullanarak paketimizi parçaladık.

Artık elimizde rotanın IP şeması, mesaj ve gönderici-alıcı bilgileri bulunuyor.

```
while True:
    connection, address = clientSocket.accept()

    if connection is not None:
        receivedMessage = connection.recv(1024)
        if receivedMessage != b'':

            decodeMessage = receivedMessage.decode("UTF-8")
            splitMessage = decodeMessage.split('|')
            splitMessage = [x for x in splitMessage if x]

            macHeader = splitMessage[0]
            ipHeader = splitMessage[1]
            routePathHeader = splitMessage[2]
            splitIPHeader = ipHeader.split(',')
            splitMACHeader = macHeader.split(',')

            print("----------------")
            print("Mesaj geldi! Detaylar:")
            print("Gönderen IP:", splitIPHeader[int(len(splitIPHeader) - 2)].split('-')[0])
            print("Gönderen MAC:", splitMACHeader[int(len(splitMACHeader) - 2)].split('-')[0])
            print("Mesaj:", splitMessage[-1])
            print("----------------")
            connection.close()
    else:
        print("Bağlantı koptu!")
```