بسم الله الرحمن الرحیم

دانشکده مهندسی برق

شبکه های مخابرات داده

استاد مربوطه : دکتر پاکروان

گزارش پروژه ی شبیه سازی AODV

نام و نام خانوادگی :

علیرضا شیرزاد

شماره دانشجویی :

95101847

تیر 1398

چند نکته ی مهم :

1 _ کد های Server و Vehicles به خاطر تفاوت ماهوی در دو py. متفاوت نوشته شده اند و باید در دو ترمینال جدا اجرا شوند .

2 _ حتی المقدور کد ها در محیط Jupyter Notebook ران شوند .

3 _ به علت زمان بر بودن عملیات وارد کردن مشخصات vehicle ها ، این داده ها را بالای فایل سناریو کپی کنید . یک نمونه از فایل سناریو همراه گزارش کار ضمیمه شده است .

5 _ اینجانب برای اولین بار در محیط برنامه نویسی پایتون کد زده ام لذا ممکن است در بسیاری از موارد بهینه ترین راه ها را انتخاب نکرده باشم .

6 _ در این گزارش به کلیات و بخش های اصلی و مهم پروژه می پردازم و از تعاریف جزئیات توابع و متغیر ها و توابع کمکی پرهیز کردم .

7 _ طبق مطالعات بنده در برخی Implementation های پروتکل AODV ، از فیلد Destination Sequence Number در پیام های RREQ و RREP استفاده نشده . در این پروژه نیز از این فیلد استفاده نکرده ام.

با تشکر

ساختار کلی Script :

این پروژه روی دو ترمینال مختلف و با دو کد py. مختلف اجرا می شود . کد اول به نام server.py وظیفه ی شبیه سازی server را دارد که و کد دوم به نام vehicles.py وظیفه ی شبیه سازی ماشین ها و در واقع Client ها را دارد .

Server :

وظیفه ی Server در پروژه ی اینجانب در دو جا حائز اهمیت است :

- ✓ جابجایی و Handover پیام ها از یک socket به socket دیگر
- ✓ شناسایی همسایه های Node ها فقط در فاز Initialization

Vehicles :

تقریبا تمام اتفاقات در کد Vehicles انجام می پذیرد . این کد شامل تمام حرکات و اطلاعات Vehicle ها و اجرای سناریو می باشد .

Receiver :

پیچیده ترین عنصر در این پروژه receiver هر vehicle است که بسته به prefix هر پیام تشخیص می دهد که پیام از چه جنسی است . انواع پیام ها در جدول زیر آمده است :

Message Types
intialization
Initialization Reply
Hello
Hello Reply
Finished Hello
Route Request
Route Reply
Data Message
Routing Table

در Reciever تصمیم گرفته می شود که بسته به نوع هر پیام چه عملیاتی اجرا شود .

```
def Receive(self):
    msg = ''
    while True:
        data = self.csocket.recv(2048)
        msg = data.decode()
        msg = mdecode (msg)
        if (msg[0:4] == 'init'):
            print("\n"+ msg)
            self.Messages.append(msg)
            temp = [int(s) for s in msg.split() if s.isdigit()]
            idx = UIDs.index(temp[0])
            x = (temp[1],temp[2])
            y = (self.LocationX,self.LocationY)
            Distance = math.sqrt(sum([(a - b) ** 2 for a, b in zip(x, y)]))
            self.DistanceTable[idx] = Distance
            self.Send('InitRep : SourceUID ' + str(self.UID) + ' Destination UID ' + str(temp[0]) )
        elif (msg[0:7] == 'InitRep'):
            self.Messages.append(msg)
            print("\n"+ msg )
        elif (msg[0:9] == '-HelloRep'):
            self.Messages.append(msg)
            print("\n" + msg )
        elif (msg[0:8]=='Finished'):
            print(msg)
            self.Messages.append(msg)
            idx = UIDs.index(self.UID)
            HelloReplyList[idx] = True
```

```
elif (msg[0:4]=='RREQ'):
    print("\n" + msg )
    temp = [int(s) for s in msg.split() if s.isdigit()]
    SrcUID = temp[0]
    DstUID = temp[1]
    Dstidx = UIDs.index(DstUID)
    Srcidx = UIDs.index(SrcUID)
    SrcSeqNum = temp[2]
    HopCount = temp[3]
    fromUID = temp[4]
    self.Messages.append("from "+ str(fromUID)+msg)
    if self.RoutingTable[Srcidx][3]<SrcSeqNum :
        self.RoutingUpdatefunc(SrcUID,fromUID,HopCount,SrcSeqNum)
        if HopCount < Max_HOPCount :
            temp = [int(s) for s in msg.split() if s.isdigit()]
            if self.RoutingTable[Dstidx][5] == True:
                self.RREP(SrcUID,DstUID)
            else:
                HopCount  = HopCount + 1
                msg = 'RREQ : SRCUID= ' + str(SrcUID) + ' DSTUID= ' + str(DstUID) + ' SrcSeqNum ' + str(SrcSeqNu
                self.Broadcast(msg,fromUID)
```

```
elif (msg[0:4]=='RREP'):
    print("\n" + msg)
    temp = [int(s) for s in msg.split() if s.isdigit()]
    print(msg)
    SrcUID = temp[0]
    DstUID = temp[1]
    Dstidx = UIDs.index(DstUID)
    Srcidx = UIDs.index(SrcUID)
    SrcSeqNum = temp[2]
    HopCount = temp[3]
    fromUID = temp[4]
    self.Messages.append("from "+ str(fromUID)+msg)
    if self.RoutingTable[Srcidx][3]<SrcSeqNum :
        self.RoutingUpdate(SrcUID,fromUID,HopCount,SrcSeqNum)
        if HopCount<Max_HOPCount :
            if self.UID == DstUID:
                print ("\nRoute has been stablished successfully !!!")
            else:
                HopCount  = HopCount + 1
                msg = 'RREP : SRCUID= ' + str(SrcUID) + ' DSTUID= ' + str(DstUID) + ' SrcSeqNum ' + str(SrcSeqNu
                self.Send(msg)
```

```
elif (msg[0:7]=='Message'):
    print("\n" + msg + "\n")
    temp = [int(s) for s in msg.split() if s.isdigit()]
    DstUID = temp[1]
    if DstUID==self.UID :
        s = msg.find("::")
        SrcUID = temp[0]
        M = msg[s+2:len(msg)-1]
        s = M.find("::")
        M = M[0:s]
        print ('\n Messgae from UID ' + str(SrcUID) + ':         " ' + M )
        end = time.time()
        print(end - start)
        self.Messages.append("from "+ str(SrcUID)+ M)
    else:
        self.Send(msg)
elif (msg[0:5]=='Hello'):
    print("\n" + msg + "  Destination UID " + str(self.UID))
    self.Messages.append(msg)
    temp = [int(s) for s in msg.split() if s.isdigit()]
    idx = UIDs.index(temp[0])
    x = (temp[1],temp[2])
    y = (self.LocationX,self.LocationY)
    Distance = math.sqrt(sum([(a - b) ** 2 for a, b in zip(x, y)]))
    self.DistanceTable[idx] = Distance
    self.Send('-HelloRep : SourceUID ' + str(self.UID) + ' Destination UID ' + str(temp[0]) )
```

:Transmitter

هر Transmitter توانایی این را دارد که تشخیص بدهد چه پیامی در حال ارسال است و با اضافه کردن Prefix مناسب ، Receiver را از نوع پیام با خبر سازد .

```
def Sendfunc(self,msg):
    if msg[0:7] == 'Message':
        temp = [int(s) for s in msg.split() if s.isdigit()]
        DstUID = temp[1]
        msg = msg + " nextHOP = " + str(self.findNextHOP(DstUID))
        time.sleep(self.Delay)
        self.csocket.sendall(bytes(code(msg),'UTF-8'))
    elif msg[0:4] == 'RREP' :
        if self.UID==12:
            print()
        temp = [int(s) for s in msg.split() if s.isdigit()]
        UID = temp[1]
        msg = msg + " nextHOP = " + str(self.findNextHOP(UID))
        time.sleep(self.Delay)
        self.csocket.sendall(bytes(code(msg),'UTF-8'))
    else :
        time.sleep(self.Delay)
        self.csocket.sendall(bytes(code(msg),'UTF-8'))
```

Classes :

دو کلاس مهم در Server و vehicles ساخته شده که عبارتند از :

1. کلاس Vehicles : این کلاس در کد vehicles ساخته شده و object های آن نمایانگر هر Vehicle است . این کلاس شامل attribute ها و function های زیر است :

ATTRIBUTES
IP
Port
UID
Location X
Location Y
Delay
Distance Table
Routing Table

FUNCTION	ARGUMENT
Receive	None
RREP	(SrcUID,DstUID)
RoutingUpdate RoutingUpdatefunc	(SrcUID,fromUID,HopCount,SrcSeqNum) (SrcUID,fromUID,HopCount,SrcSeqNum)
Send Sendfunc	(msg) (msg)
Broadcast Broadcastfunc	(msg,fromUID) (msg,fromUID)
findNextHOP	(SrcUID)
Listen	None
init initialize	None None
RREQ	(DstUID)
Hello SendHello	None None

توابعی که برای اجرا نیاز به threading داشتند شامل دو تابع می باشند . یک تابع برای ساختن یک thread و راه اندازی thread تابع و تابع دیگر که عملیات اصلی در آن اجرا می شود . این توابع در جدول فوق زیر هم و بدون فاصله درج شده اند .

2. کلاس ClientThreead : این کلاس در کد Server ساخته می شود و object های آن نمود حقیقی ندارند و صرفا عملیات Handover را بین Socket ها انجام می دهند . Attribute ها و Function های این کلاس به صورت زیر می باشد :

ATTRIBUTES
Client Address
Client Socket
Client Distance Table
Client Location X
Client Location Y
Client UID

FUNCTION	ARGUMENT
Receive	None
Send	(msg)
Sendfunc	(msg)
Run	None

Initialization phase :

در ابتدا یک Socket برای Server می سازیم و سپس منتظر Connect شدن Client ها می مانیم :

```
#Creating the server socket
LOCALHOST = "127.0.0.1"
PORT = 8081
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((LOCALHOST, PORT))
print("\nServer started\n")
print("\nWaiting for client request..\n")
Clients = []

# Connecting the servers to vehicles
for i in range(N):
    server.listen()
    clientsock, clientAddress = server.accept()
    newClienthread = ClientThread(clientAddress, clientsock,r)
    newClienthread.run()
    Clients.append(newClienthread)
```

پس از Connect شدن هر یک از vehicle ها در سمت Server برای هر کدام یک Object از کلاس ClientThread می سازیم . از طرفی در سمت Client داریم :

```
# Vehicle Creation
V = []
Delays = []
for i in range(N):
    temp = [int(s) for s in lines[i+7].split() if s.isdigit()]
    v = Vehicle(temp[0],temp[1],temp[2],temp[3],temp[4],temp[5],temp[6])
    V.append(v)
    V[i].Listen()
    V[i].init()
    Delays.append(temp[6])
```

که به تعداد مشخص شده در سناریو ، از کلاس Vehicle ، Object می سازد . هر vehicle پس از ساخته شدن دو کار اصلی می کند :

1. راه اندازی Listenin Thread برای نصب یک گیرنده ی دائمی در آن vehicle
2. شروع عملیات initialization

در شروع عملیات Initialization هر vehicle از موقعیت جغرافیایی vehicle های دیگر خبر ندارد لذا یکی پیام init به server ارسال می کند و موقعیت خود را اعلام می کند . سپس Server همسایه های هر vehicle را تشخیص می دهد و پیام initializaion را به آن ها Forward می کند . پس از اینکه یک vehicle پیام init را دریافت کرد یک پیام initreply را به سمت همسایه های خود ارسال می کند و دو لیست مهم را در attribute های خود update می کند :

1. لیست Distance : هر vehicle فاصله ی خود از دیگر vehicle ها را در آن ذخیره می کند . برای vehicle هایی که همسایه نیستند مقدار بی نهایت (Maximum Distance) را قرار می دهد .
2. لیست Routing Table : این لیست اساس تصمیم گیری هر Vehicle است.فرمت Routing Table به صورت زیر است :

Destination UID	Next Hop	Hop Count	Sequence Number	Validity Flag
...	...	...	...	...

هر Vehicle در مرحله ی Initialization همه ی Validity Flag ها را False می کند به غیر از ردیف های مربوط به همسایه های خود که به آن ها مسیر دارد .

در انتهای مرحله ی Initialization همه ی vehicle ها به اندازه ی ماکزیمم تاخیر vehicle های صبر می کنند تا همه ی Vehicle ها RoutingTable خود را Update کرده باشند . سپس سناریو خط به خط اجرا می شود .

Scenario:

پس از مرحله ی initialization همه ی vehicle ها آماده ی اجرای سناریو هستند . دستورات سناریو به سه گونه هستند :

1. Send Message
2. Change Location
3. Wait

نمونه ای از سناریو به شکل زیر است :

```
%% Field Parameters
Vehicle Diameter = 20
Field Length = 70
Field width = 70
Number of vehicles = 10
ExpirationTime = 7
%%
1 127.0.0.2 3001 0 10 3
2 127.0.0.3 3002 3 2 1
3 127.0.0.4 3003 0 22 1
4 127.0.0.5 3004 6 33 2
5 127.0.0.6 3005 9 41 1
6 127.0.0.7 3006 4 48 1
7 127.0.0.8 3007 12 63 3
8 127.0.0.9 3008 12 56 3
9 127.0.0.10 3009 22 10 1
10 127.0.0.11 3010 34 5 3
%%
ChangeLoc 1-0-10 2-3-2 3-0-22 4-6-33 5-9-41 6-4-48
SendMessage 20-hello-4
```

Send Message:

پس از این دستور vehicle مبدا ، ردیف UID مقصد را در Routing Table خود چک می کند . اگر Validity Flag مربوطه True بود پیام را از طریق Server به next Hop ارسال می کند .

```
def SendMessage(SrcUID,DstUID,msg):
    msg = 'Message from ' + str(SrcUID) + ' to ' + str(DstUID) + ' ::' + msg + ':: '
    idx = UIDs.index(SrcUID)
    V[idx].Send(msg)
```

تابع findnextHOP فراخوانده می شود و HOP بعدی False بود عملیات Route Request آغاز می شود .

```
def findNextHOP(self,SrcUID) :
    idx = UIDs.index(SrcUID)
    if self.RoutingTable[idx][5]:
        return self.RoutingTable[idx][1]
    else :
        self.RREQ(SrcUID)
        while self.RoutingTable[idx][5] == False :
            pass
        return self.RoutingTable[idx][1]
```

در عملیات Route Request تابع find next Hop در یک Dummy While منتظر می مانند تا عملیات تمام شود . همچنین بسته ی Route Request به شکل Flooding در شبکه پخش می شود .

```
def RREQ(self,DstUID) :
    idx = UIDs.index(self.UID)
    self.RoutingTable[idx][3] += 1
    msg = 'RREQ : SRCUID= ' + str(self.UID) + ' DSTUID= ' + str(DstUID) + ' SrcSeqNum ' + str(self.RoutingTable
    self.Broadcast(msg,0)
```

پس از رسیدن پیام RRQ به vehicle ای که مسیر معتبری به vehicle مقصدر دارد پیام RREP تولید شده و به سمت مقصد Unicast می شود . زمانی RREP به vehicle مبدا می رسد Routing Table به روز رسانی شده و findNextHOP آدرس HOP بعدی را به Transmitter می دهد و پیام ارسال می شود .

Change Location:

این دستور در ابتدا Location ها را طبق فرمت داده شده تغییر می دهد و سپس تمام Routing Table های تمام vehicle ها را نامعتبر(False) می کند . سپس عملیات Hello را شروع می کند .

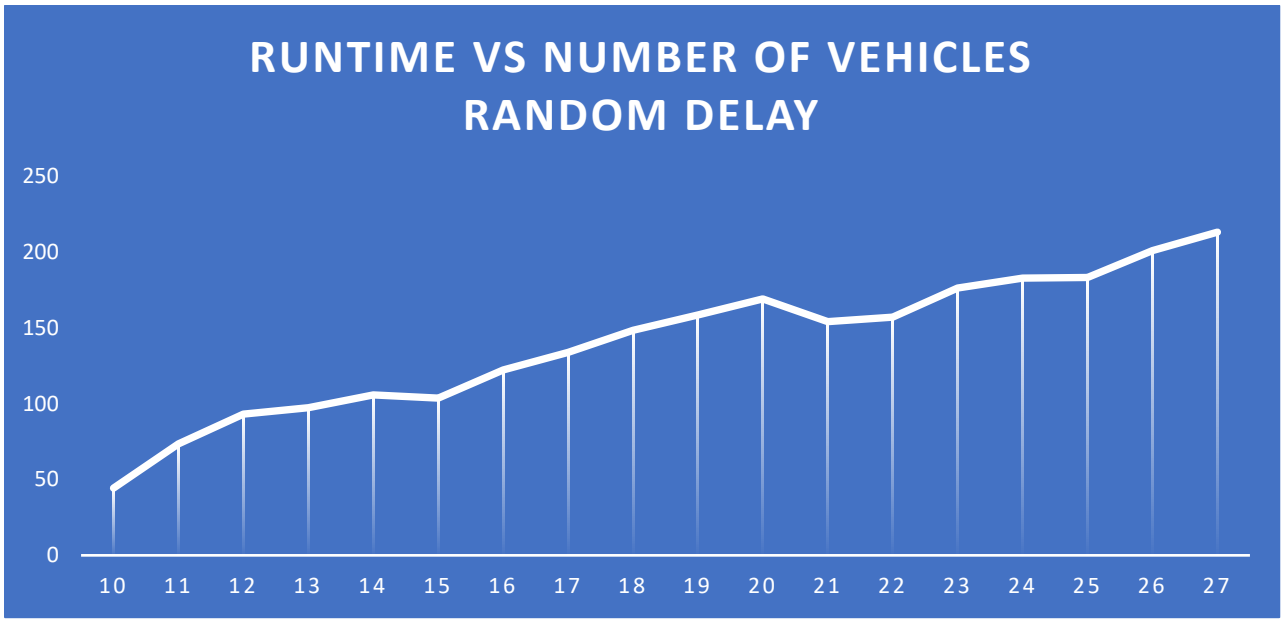
```
def ChangeLoc(uids,xs,ys):
    for i in range(N) :
        HelloReplyList[i] = False
    CNums = len(uids)
    for p in range(CNums):
        idx = UIDs.index(uids[p])
        V[idx].LocationX = xs[p]
        V[idx].LocationY = ys[p]
    for i in range(N):
        V[i].DistanceTable = [Max_Distance] * N
        V[i].RoutingTable[:][1:5] = [[] for x in range(5) for y in range(1)]
        for j in range(N):
            V[i].RoutingTable[j][5] = False
        V[i].Hello()
    q = [True]*N


    while HelloReplyList != q:
        pass

    for i in range(N):
        V[i].DistanceTable[i] = 0
        for j in range(N):
            V[i].RoutingTable[j][3] = 0
            V[i].RoutingTable[j][0] = V[j].UID
            if V[i].DistanceTable[j] < V[i].diameter :
                V[i].RoutingTable[j][5] = True
                V[i].RoutingTable[j][1] = V[j].UID
            else:
                V[i].RoutingTable[j][5] = False
```

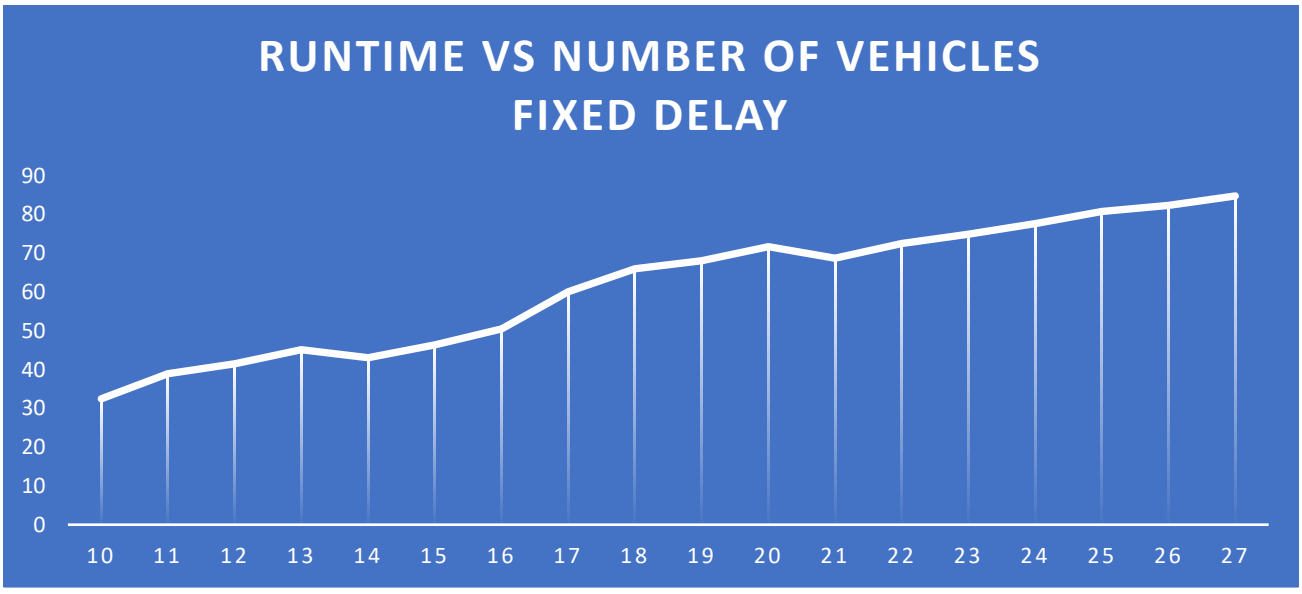
پس از اجرای عملیات Hello ، Vehicle ها اندیس مربوط به خود را در لیست HelloReplyList ، False می کنند بدین معنی که هنوز vehicle مربوطه Hello ها را از همسایه های جدید خودش دریافت نکرده لذا هیچ Scenario ای حق اجرا شدن ندارد . با شروع عملیات Hello ، هر Vehicle پیام Hello را به Server تحویل می دهد و Location جدید خود را به آن اعلام می کند . سپس Server به مانند عملیات Initialization ، پیام را برای همسایه های آن vehicle فروارد می کند . همچنین وقتی Hello Packet ها را به یک vehicle ارسال کرد اگر دیگر همسایه ای ندارد برای آن Vehicle پیام Finished Hello را ارسال می کند بدین معنی که vehicle مورد نظر اطلاعات لازم را درباره ی همسایه ها ی خود بدست آورده و دیگر منتظر HelloPacket نباشد . وقتی vehicle پیام Finished Hello را دریافت می کند خانه ی مربوط به خود را در لیست HelloReplyList ، True می کند. زمانی که تمام لیست True شده باشد یعنی همه ی Vehicle ها از همسایه های جدید خود باخبر شده اند و Scenario اجازه ی از سرگیری را دارد .

نمودار زمان Runtime ارسال پیام برای یک توپولوژی خطی بر حسب تعداد Vehicle ها با تاخیر های Random به شکل زیر است :



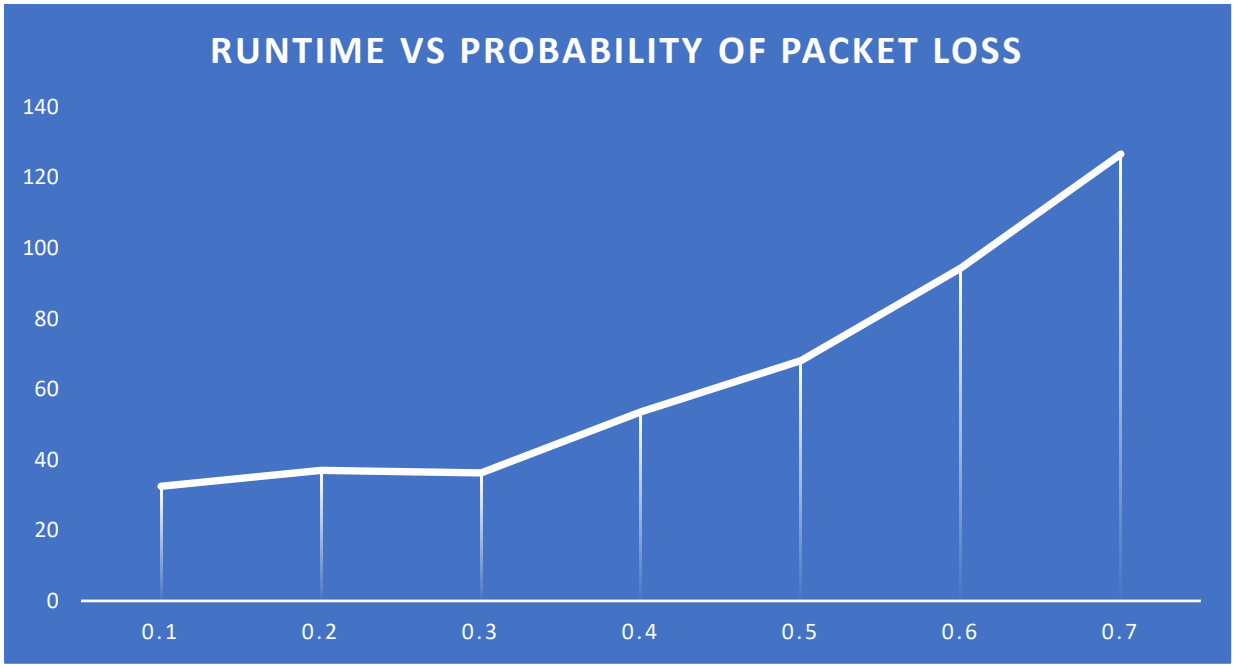
مشاهده می کنیم که با افزایش تعداد Vehicle های میزان زمان ارسال تقریبا به صورت خطی افزایش می یابد. نقاطی از نمودار که نمودار نزولی شده به خاطر این است که در آن زمان Kernel خود برای ژوپیتر را Reset کردم لذا Runtime کمی کاهش یافت . ولی در کل خطی بودن رفتار کاملا مشهود است .

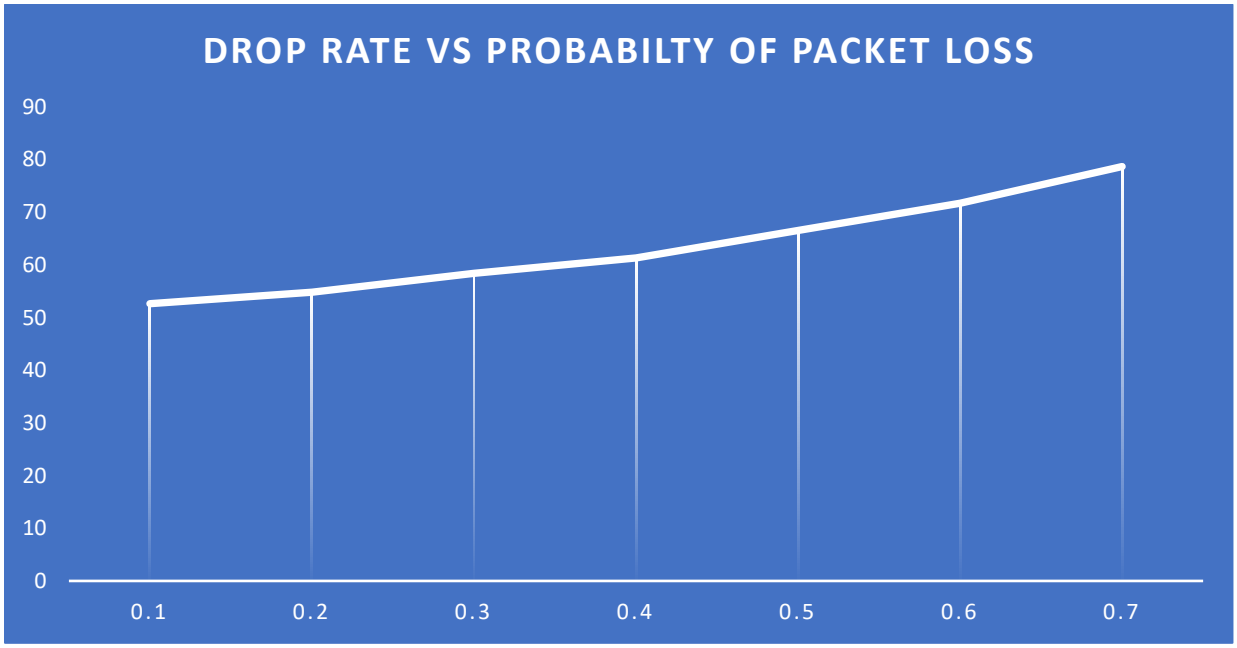
نمودار زمان Runtime ارسال پیام برای یک توپولوژی خطی بر حسب تعداد Vehicle ها با تاخیر های Fixed (1sec) به شکل زیر است :



که شباهت بسیار زیادی با حالت Random دارد و جفت حالت خطی دارند .

هم چنین در server تابعی طراحی می کنیم که با احتمال p ، packet ها را drop کند . نتیجه ی امر به شکل زیر می باشد :





مشاهده می کنیم که زمان ارسال پیام بسیار شدید و به صورت نمایی با احتمال خطا بالا می رود در صورتی که نرخ Packet Drop تقریبا خطی است .

توابع فرعی:

1. یکی از مشکلاتی که در این پروژه با آن مواجه شدم این بود که پیام قبلی روی Socket به پیام بعدی اضافه می شد و یک پیام خراب می شد . برای حل این مشکل ابتدا و انتهای پیام را در گیرنده مشخص و در فرستنده آن را بازیابی کردم . برای ان کار از دو تابع code و mdecode استفاده کردم.
2. از دیگر مشکلات پروژه این بود که با بالا رفتن Load شبکه بعضی از پیام ها به مقصد نمی رسیدند . برای حل این مشکل در زمان هایی که شبکه در حال Update شدن است و حجم ترافیک بسیار بالاست به شبکه به اندازه ی دو برابر ماکزیمم تاخیر vehicle ها تاخیر وارد کردم تا از Load شبکه کاسته شود و اطلاعات در شبکه Converge کند .

Logging:

نمونه ای از Log گرفتن کد برای یک تو پولوژی خطی 4 تایی به شکل زیر است :

```
Send 1 init 1 XLocation 0 YLocation 0
 Send 2 init 2 XLocation 0 YLocation 2
 Send 3 init 3 XLocation 0 YLocation 6
 Send 4 init 4 XLocation 6 YLocation 9
 Receive 2 init : SourceUID 1 XLocation 0 YLocation 0
 Receive 3 init : SourceUID 1 XLocation 0 YLocation 0
 Receive 1 init : SourceUID 2 XLocation 0 YLocation 2
 Receive 3 init : SourceUID 2 XLocation 0 YLocation 2
 Receive 1 init : SourceUID 3 XLocation 0 YLocation 6
 Receive 2 init : SourceUID 3 XLocation 0 YLocation 6
 Receive 4 init : SourceUID 3 XLocation 0 YLocation 6
 Receive 3 init : SourceUID 4 XLocation 6 YLocation 9
 Send 2 InitRep : SourceUID 2 Destination UID 1
 Receive 1 InitRep : SourceUID 2 Destination UID 1
```

```
Send 3 InitRep : SourceUID 3 Destination UID 1

Send 3 InitRep : SourceUID 3 Destination UID 2

Receive 1 InitRep : SourceUID 3 Destination UID 1

Update 1

Update 2

Update 3

Update 4

Receive 2 InitRep : SourceUID 3 Destination UID 2

Send 2 InitRep : SourceUID 2 Destination UID 3

Receive 3 InitRep : SourceUID 2 Destination UID 3
Send 3 InitRep : SourceUID 3 Destination UID 4


Receive 4 InitRep : SourceUID 3 Destination UID 4

Send 4 InitRep : SourceUID 4 Destination UID 3

Send 1 InitRep : SourceUID 1 Destination UID 2

Receive 2 InitRep : SourceUID 1 Destination UID 2

Send 1 InitRep : SourceUID 1 Destination UID 3

Receive 3 InitRep : SourceUID 1 Destination UID 3

Receive 2 RREQ : SRCUID= 1 DSTUID= 4 SrcSeqNum 1 HOPCount 0  from UID 1

Update 2

Receive 3 InitRep : SourceUID 4 Destination UID 3

Receive 3 RREQ : SRCUID= 1 DSTUID= 4 SrcSeqNum 1 HOPCount 1  from UID 2

Update 3

Send 3 RREP : SRCUID= 4 DSTUID= 1 SrcSeqNum 1 HOPCount 0  nextHOP = 2

Receive 3 RREQ : SRCUID= 1 DSTUID= 4 SrcSeqNum 1 HOPCount 0  from UID 1

Receive 2 RREP : SRCUID= 4 DSTUID= 1 SrcSeqNum 1 HOPCount 0  from UID 3

Update 2

Send 2 RREP : SRCUID= 4 DSTUID= 1 SrcSeqNum 1 HOPCount 1  nextHOP = 1

Receive 1 RREP : SRCUID= 4 DSTUID= 1 SrcSeqNum 1 HOPCount 1  from UID 2

Update 1
```

```
Route has been stablished successfully !!!

 Send 1 Message from 1 to 4 ::hello::  nextHOP = 2

 Receive 2 Message from 1 to 4 ::hello::

 Send 2 Message from 1 to 4 ::hello::  nextHOP = 3

 Receive 3 Message from 1 to 4 ::hello::

 Send 3 Message from 1 to 4 ::hello::  nextHOP = 4

 Receive 4 Message from 1 to 4 ::hello::
19.500836610794067
```

که خط آخر آن زمان اجرای کل برنامه است .