Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проектна задача по предметот Безжични мултимедиски системи

Изработил: Бојана Бошкова
Број на индекс: 173220

Скопје, 2020

Содржина

SimPy базиран WSN Simulator

WsnSimPy е симулатор за безжични сензорски мрежи, кој се користи за моделирање на размена на пораки на мрежно ниво и на full-stack комуникации. Напишан е во Python 3 и е изграден врз SimPy симулацискатa рамка.

Авторот на пакетот во рамки на gitlab репозиториумот [1] има изработено два примери: flooding протокол [2] и aodv протокол [3]. Во рамки на проектната задача ќе биде објаснет детално изворниот код од двата примери и соодветно излезот.

Инсталација на WsnSimpy пакетот

За потребите на проектната задача, WsnSimpy пакетот е инсталиран преку pip менаџерот изолирано во виртуелна околина. Командите за креирање на виртуелна околина, нејзино активирање и инсталација на пакетот се следните:

```
pip install virtualenv
cd BMS_Proekt
virtualenv venv
venv\Scripts\activate
(venv) pip install wsnsimpy
(venv) pip freeze
```

```
   simpy==4.0.1
   wsnsimpy==0.2.5
```

Извршување на примерите

Примерите кои се дел од пакетот се наоѓаат во директориумот wsnsimpy/examples и можат директно да се стартуваат преку командна линија (command prompt) преку следните две команди:

```
(venv) python -m wsnsimpy.examples.flood
(venv) python -m wsnsimpy.examples.aodv
```

Објаснување на изворниот код на примерите

Flood протокол

Најпрво се креира објект за симулаторот преку конструктор со именувани аргументи за секое од наведените својства **(until** - време на извршување на симулацијата (во случајот 100сек), **timescale**, **visual, terrain_size** - резолуција на прозорецот во единица мерка пиксели w x h и title - име на прозорец-от на GUI - то) на симулатор објектот. Потоа со вгнездени for наредби се иницијализираат и додаваат вкупно 100-те јазли на симулаторот. Со помош на random генераторот на целобројни вредности на интервалот [-20, 20] се задаваат позициите/координатите на торката **(px, py)** за секој јазел поединечно. За стартување на симулаторот се користи **run()** ф-јата.

```
sim = wsp.Simulator(
        until=100,
        timescale=1,
        visual=True,
        terrain_size=(700,700),
        title="Flooding Demo")
for x in range(10):
    for y in range(10):
        px = 50 + x*60 + random.uniform(-20,20)
        py = 50 + y*60 + random.uniform(-20,20)
        node = sim.add_node(MyNode, (px,py))
        node.tx_range = 75
        node.logging = True
sim.run()
```

Дефинираната класа за јазлите на симулаторот **MyNode**, наследува од **wsp.Node**, составена е од неколку својства меѓу кои најзначајни се: **recv** - boolean вредност која означува дали даденият јазел е посетен, **tx_range** - опсег/дистанца на дофат при broadcast-от, scene - визуелен приказ на јазелот и.т.н. Методата **init** со аргумент **self** референцата на тековниот јазол е конструкторот на класата во чие што тело се наоѓаат наредбите за иницијализација најпрво на super класата MyNode и задавањето на False вредноста на recv својството. Класата MyNode е составена од трите најзначајни преоптоварени нативни методи неопходни за целосното извршување на симулаторот: **run**, **broadcast** и **on_receive**. Во рамки на трите функции се користи **logger**, чии што записи можеме да го видиме во

терминал за полесно разбирање/толкување на извршувањето на симулацијата, односно редоследот на посетувањето на јазлите. **Функцијата broadcast**, како што самото име кажува, служи за активирање/повикување на најблиските јазли од даденият повикувачки јазол на растојание tx_range. **Функцијата on_receive** како влезни аргументи ги прима двата јазли: тековниот **примач-јазол** и **повикувачот-јазол**. Доколку е веќе посетен повиканиот јазол се испишува log во командна линија со следната содржина: **"Message seen; reject"** и се прекинува извршувањето на фукцијата. Во спротивно, доколку првпат се посетува јазолот, се испишува следната порака: **“New message; prepare to rebroadcast"**, се сетира recv знаменцето на True, се менува бојата во црвена на посетениот тековен јазел и повторно после одредено време timeout се врши broadcast од тековно(последно) посетениот јазол во улога на sender (јазол-повикувач). **Функцијата run** се извршува на самиот почеток при стартувањето на симулаторот, односно секој јазел се проверува дали е изворниот јазол на почеток на симулацијата (најгоре во изворниот код е дефинирана променлива **SOURCE = 35** - id/реден број на јазолот во опсег од 1 до 100). За изворниот јазол се извршуваат следните инструкции: се означува јазелот со сива боја, се сетира знаменцето recv = True за посетеност и после timeout од две милисекунди започнува broadcast-от од изворниот јазол, додека сите останати јазли се маркираат со сива боја.

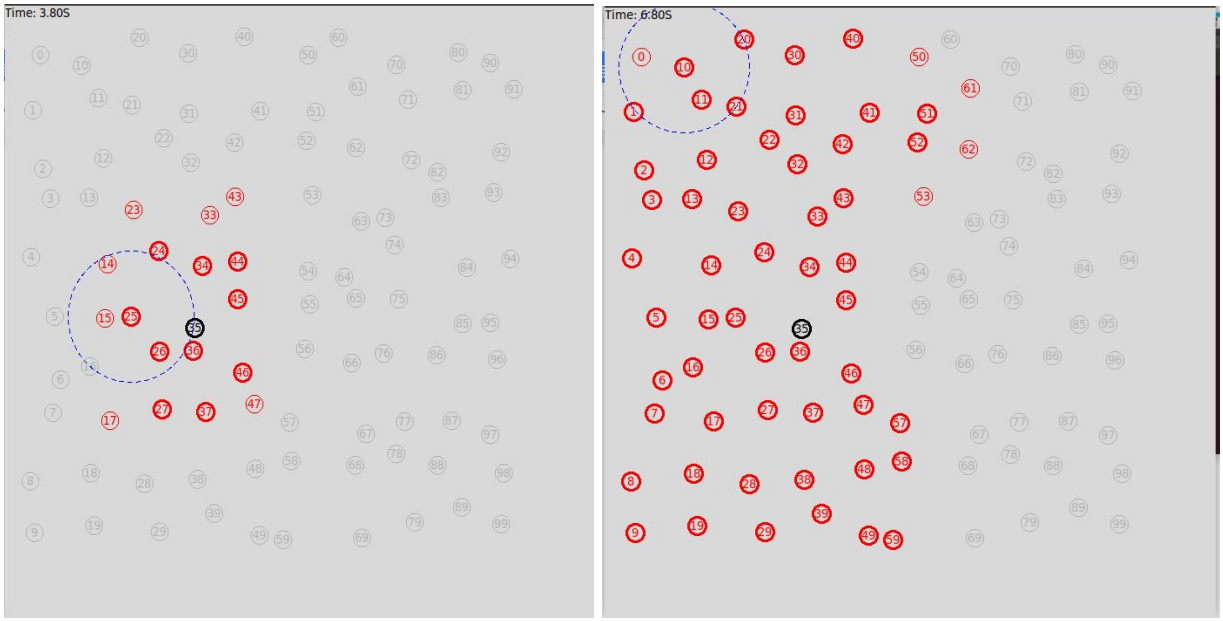
```
class MyNode(wsp.Node):
    tx_range = 100

    def init(self):
        super().init()
        self.recv = False

    def run(self):
        if self.id == SOURCE:
            self.scene.nodecolor(self.id,0,0,0)
            self.recv = True
            yield self.timeout(2)
            self.broadcast()
        else:
            self.scene.nodecolor(self.id,.7,.7,.7)

    def broadcast(self):
        self.scene.nodewidth(self.id, 3)
        self.log(f"Broadcast message")
        self.send(wsp.BROADCAST_ADDR)
    def on_receive(self, sender, **kwargs):
        self.log(f"Receive message from {sender}")
```

```
        if self.recv:
            self.log(f"Message seen; reject")
            return
        self.log(f"New message; prepare to rebroadcast")
        self.recv = True
        self.scene.nodecolor(self.id,1,0,0)
        yield self.timeout(random.uniform(0.5,1.0))
        self.broadcast()
```

Логови од извршувањето на flood example-от до 3.56 секунда:

```
Node #35 [   2.00000] Broadcast message
Node #36 [   2.00003] Receive message from 35
Node #36 [   2.00003] New message; prepare to rebroadcast
Node #26 [   2.00005] Receive message from 35
Node #26 [   2.00005] New message; prepare to rebroadcast
Node #45 [   2.00006] Receive message from 35
Node #45 [   2.00006] New message; prepare to rebroadcast
Node #34 [   2.00007] Receive message from 35
Node #34 [   2.00007] New message; prepare to rebroadcast
Node #45 [   2.50436] Broadcast message
Node #44 [   2.50441] Receive message from 45
Node #44 [   2.50441] New message; prepare to rebroadcast
Node #34 [   2.50442] Receive message from 45
Node #34 [   2.50442] Message seen; reject
Node #35 [   2.50442] Receive message from 45
Node #35 [   2.50442] Message seen; reject
Node #36 [   2.63688] Broadcast message
Node #35 [   2.63690] Receive message from 36
Node #35 [   2.63690] Message seen; reject
Node #26 [   2.63692] Receive message from 36
Node #26 [   2.63692] Message seen; reject
Node #46 [   2.63694] Receive message from 36
Node #46 [   2.63694] New message; prepare to rebroadcast
Node #37 [   2.63695] Receive message from 36
Node #37 [   2.63695] New message; prepare to rebroadcast
Node #26 [   2.82994] Broadcast message
Node #36 [   2.82998] Receive message from 26
Node #36 [   2.82998] Message seen; reject
Node #35 [   2.82999] Receive message from 26
Node #35 [   2.82999] Message seen; reject
Node #25 [   2.82999] Receive message from 26
Node #25 [   2.82999] New message; prepare to rebroadcast
Node #27 [   2.83001] Receive message from 26
Node #27 [   2.83001] New message; prepare to rebroadcast
Node #34 [   2.94729] Broadcast message
Node #44 [   2.94733] Receive message from 34
Node #44 [   2.94733] Message seen; reject
Node #24 [   2.94734] Receive message from 34
```

```
Node #24 [   2.94734] New message; prepare to rebroadcast
Node #45 [   2.94735] Receive message from 34
Node #45 [   2.94735] Message seen; reject
Node #33 [   2.94735] Receive message from 34
Node #33 [   2.94735] New message; prepare to rebroadcast
Node #35 [   2.94736] Receive message from 34
Node #35 [   2.94736] Message seen; reject
Node #27 [   3.39348] Broadcast message
Node #37 [   3.39353] Receive message from 27
Node #37 [   3.39353] Message seen; reject
Node #17 [   3.39355] Receive message from 27
Node #17 [   3.39355] New message; prepare to rebroadcast
Node #26 [   3.39355] Receive message from 27
Node #26 [   3.39355] Message seen; reject
Node #44 [   3.47533] Broadcast message
Node #34 [   3.47537] Receive message from 44
Node #34 [   3.47537] Message seen; reject
Node #45 [   3.47537] Receive message from 44
Node #45 [   3.47537] Message seen; reject
Node #33 [   3.47539] Receive message from 44
Node #33 [   3.47539] Message seen; reject
Node #43 [   3.47540] Receive message from 44
Node #43 [   3.47540] New message; prepare to rebroadcast
Node #46 [   3.48611] Broadcast message
Node #47 [   3.48615] Receive message from 46
Node #47 [   3.48615] New message; prepare to rebroadcast
Node #37 [   3.48618] Receive message from 46
Node #37 [   3.48618] Message seen; reject
Node #36 [   3.48618] Receive message from 46
Node #36 [   3.48618] Message seen; reject
Node #24 [   3.55373] Broadcast message
Node #34 [   3.55379] Receive message from 24
Node #34 [   3.55379] Message seen; reject
Node #23 [   3.55379] Receive message from 24
Node #23 [   3.55379] New message; prepare to rebroadcast
Node #14 [   3.55380] Receive message from 24
Node #14 [   3.55380] New message; prepare to rebroadcast
Node #33 [   3.55381] Receive message from 24
Node #33 [   3.55381] Message seen; reject
Node #37 [   3.56045] Broadcast message
Node #27 [   3.56050] Receive message from 37
Node #27 [   3.56050] Message seen; reject
Node #47 [   3.56051] Receive message from 37
Node #47 [   3.56051] Message seen; reject
Node #46 [   3.56051] Receive message from 37
Node #46 [   3.56051] Message seen; reject
Node #36 [   3.56052] Receive message from 37
Node #36 [   3.56052] Message seen; reject
```



Слики број 1.1, 1.2 - Screenshots од GUI на симулацијата flood со траење до 6.8 секунда

AODV протокол

Процесот на иницијализација на симулатор објектот и додавањето на јазлите на симулаторот преку вгнездените for - циклуси е аналоген на претходниот пример. Во овој пример, дополнително во визуелниот приказ покрај промената на бојата на јазлите преку командата **`self.scene.nodecolor(self.id,0,.7,0)`**, додадена е инструкција за задавање/стилизирање на parent линковите помеѓу јазлите преку командата **`sim.scene.linestyle("parent",color=(0,.8,0), arrow="tail", width=2).`** Во претходниот пример имаше дефинирано само **SOURCE id** дефинирана променлива, додека во овој пример дополнително за објаснување на работата на протоколот воведуваме променлива **DEST-инаџиско id** на јазол = 99.

```
SOURCE = 1
DEST   = 99

sim = wsp.Simulator(
        until=100,
        timescale=1,
        visual=True,
        terrain_size=(700,700),
        title="AODV Demo")

# define a line style for parent links
```

```
sim.scene.linestyle("parent", color=(0,.8,0), arrow="tail", width=2)

# place nodes over 100x100 grids
for x in range(10):
    for y in range(10):
        px = 50 + x*60 + random.uniform(-20,20)
        py = 50 + y*60 + random.uniform(-20,20)
        node = sim.add_node(MyNode, (px,py))
        node.tx_range = 75
        node.logging = True

# start the simulation
sim.run()
```

Класата за дефинирање на јазлите MyNode содржи конструктор **init метода**, која што најпрво врши иницијализација на **супер класата wsp.Node**, а потоа и задавање на вредноста на **prev својството** на None, кое што не беше присутно во претходниот пример (таму се менуваше вредноста на знаменцето recv на секој јазол). Главната разликата при работата на класата MyNode е при дефинициите на членовите функции. Класата MyNode е составена од **6 методи** заедно со init методата објаснета погоре севкупно 7 и тоа: **`run, send_rreq, send_rreply, start_send_data, send_data и on_receive.`** Методите **`on_receive`** и **`run`** се спомнати и во претходниот пример. Како што напоменав, **функцијата run** се извршува на почеток на симулацијата, односно за секој јазел според неговото id се проверува **дали е изворен** (постои само еден изворен јазол на кој му се задава сина боја и големина 2 единици, односно после timeout од 1msec се повикува функцијата **`send_rreq`** со аргумент id-то на тековниот јазол-повикувач. Во позадина на извршувањето на ф-јата **`send_rreq`** се прави broadcast до сите најблиски јазли во опсег **`tx_range`** од јазолот повикувач со тип на порака **`msg='rreq'`**), **дали е дестинациски** (постои само еден дестинациски јазол на кој му се задава црвена боја и големина 2 единици) или останат тип на меѓу јазол (повеќе јазли на кои им се задава сива боја). За најдобро да се објасни работа на симулаторот за AODV протоколот, потребно е да се објасни **изворниот код на `on_receive` методата**. Методата на влез прима 5 значајни аргументи: **тековен јазел-примач**, **претходен јазел испраќач**, **тип на порака**, **изворен јазол**. Типот на порака msg што се прима има од една од следните три вредности: **`'rreq', 'rreply' и 'data'`**. Во зависност од типот на порака што се прима се извршува одреден дел од кодот карактеристичен за тој условен случај. Доколку типот на порака **`(msg == 'rreq')`**, најпрво се проверува дали тековниот јазел-примач има претходник, во тој случај престанува со извршување методата преку **`return`**

наредбата. Во спротивно, доколку тековниот јазол-примач нема претходник, за негов претходник се доделува јазелот-испраќач на пораката, се додава врска/линк помеѓу јазелот-испраќач и тековниот јазел-примач од тип “parent”. Следно се проверува дали тековниот примач-јазел е дестинацискиот јазел, доколку е исполнет условот во конзола се испишува следната порака: **`f"Receive RREQ from {src}"`** и се повикува функцијата **`send_rreply`** со **аргумент тековниот дестинациски јазел**. Во рамки на кодот на функцијата **`send_rreply`** кој што **прима два аргументи: тековен јазел-примач на одговорот и изворен јазел-праќач**, се проверува дали се работи за дестинацискиот јазол. Во тој случај се сетира бојата на јазолот и неговата големина, а во спротивно се проследува одговорот rreply пораката до претходникот (родителот на тековниот јазел). Доколку типот на порака **`(msg == ‘rreply’)`**, за следбеник на тековниот јазол-примач на одговорот се задава sender-от јазелот праќач на пораката. Кога ќе се врати одговорот се до изворниот јазел **`(self.id is SOURCE)`** се испишува во конзола следната порака: **`f"Receive RREP from {src}"`**, се чека timeout од 5msec, па се логира одново нова порака во конзола: **`"Start sending data"`** и за крај се повикува функцијата **`self.start_process(self.start_send_data())`** со што започнува процесот на испраќање на нумерирани инкрементали секцвенци на податови од изворниот до дестинацискиот јазол преку скок од еден чекор (**`self.next`**) со повикување на помошната функција со **`send_data`** со следните три аргументи: **тековен јазол-примач, јазол-испраќач и seq - id-то на секвенцата што се испраќа.**

```
class MyNode(wsp.Node):
    tx_range = 100

    def init(self):
        super().init()
        self.prev = None

    def run(self):
        if self.id is SOURCE:
            self.scene.nodecolor(self.id,0,0,1)
            self.scene.nodewidth(self.id,2)
            yield self.timeout(1)
            self.send_rreq(self.id)
        elif self.id is DEST:
            self.scene.nodecolor(self.id,1,0,0)
            self.scene.nodewidth(self.id,2)
        else:
            self.scene.nodecolor(self.id,.7,.7,.7)
```

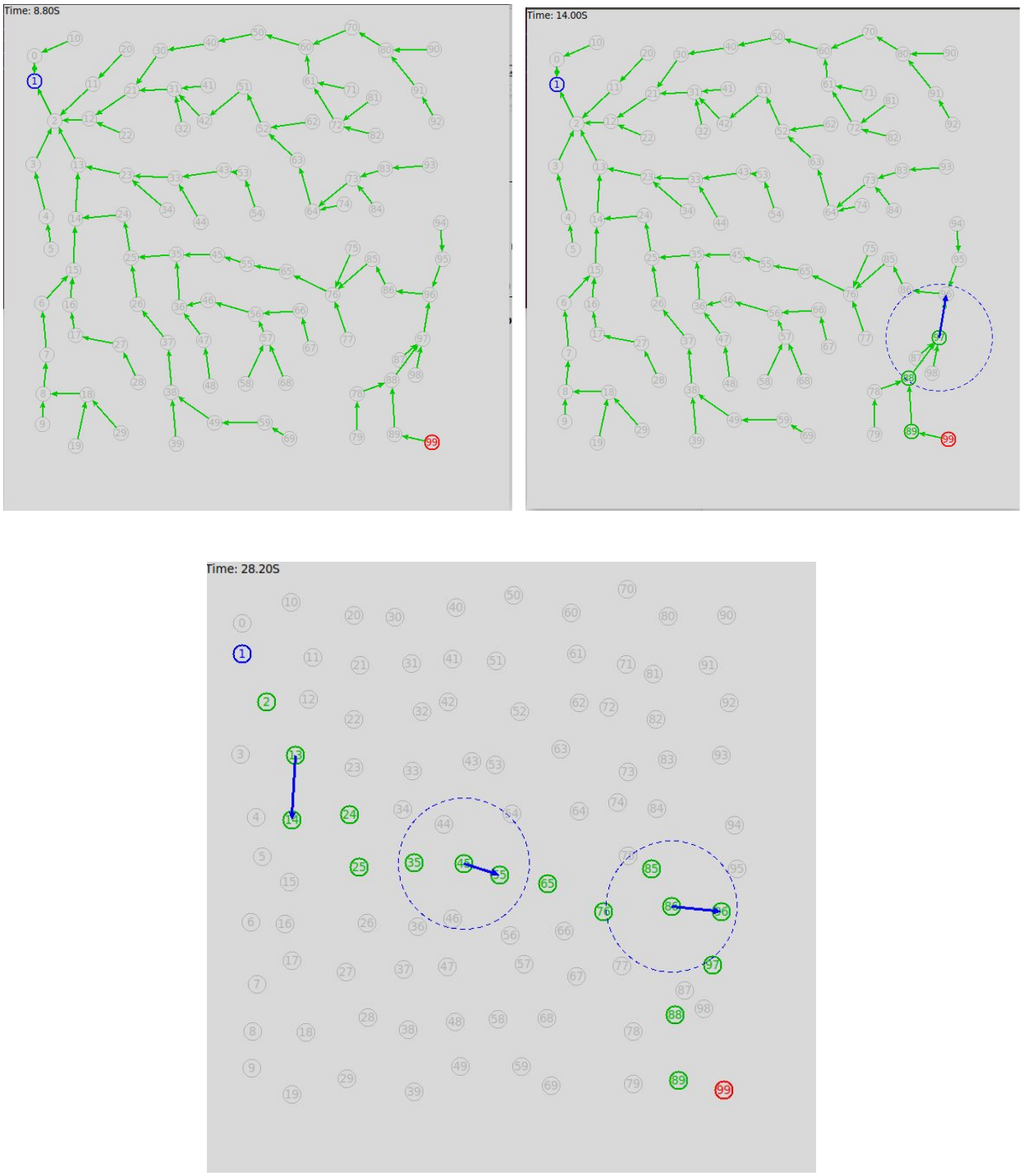
```
    def send_rreq(self,src):
        self.send(wsp.BROADCAST_ADDR, msg='rreq', src=src)

    def send_rreply(self,src):
        if self.id is not DEST:
            self.scene.nodecolor(self.id,0,.7,0)
            self.scene.nodewidth(self.id,2)
        self.send(self.prev, msg='rreply', src=src)

    def start_send_data(self):
        self.scene.clearlinks()
        seq = 0
        while True:
            yield self.timeout(1)
            self.log(f"Send data to {DEST} with seq {seq}")
            self.send_data(self.id, seq)
            seq += 1

    def send_data(self,src,seq):
        self.log(f"Forward data with seq {seq} via {self.next}")
        self.send(self.next, msg='data', src=src, seq=seq)

    def on_receive(self, sender, msg, src, **kwargs):

        if msg == 'rreq':
            if self.prev is not None: return
            self.prev = sender
            self.scene.addlink(sender,self.id,"parent")
            if self.id is DEST:
                self.log(f"Receive RREQ from {src}")
                yield self.timeout(5)
                self.log(f"Send RREP to {src}")
                self.send_rreply(self.id)
            else:
                yield self.timeout(delay())
                self.send_rreq(src)

        elif msg == 'rreply':
            self.next = sender
            if self.id is SOURCE:
                self.log(f"Receive RREP from {src}")
                yield self.timeout(5)
                self.log("Start sending data")
                self.start_process(self.start_send_data())
            else:
                yield self.timeout(.2)
                self.send_rreply(src)
```

```
        elif msg == 'data':
            if self.id is not DEST:
                yield self.timeout(.2)
                self.send_data(src,**kwargs)
            else:
                seq = kwargs['seq']
                self.log(f"Got data from {src} with seq {seq}")
```

Логови од извршувањето на AODV example-от до 26.8 секунди:

```
Node #99 [   8.41510] Receive RREQ from 1
Node #99 [  13.41510] Send RREP to 1
Node #1  [  16.61617] Receive RREP from 99
Node #1  [  21.61617] Start sending data
Node #1  [  22.61617] Send data to 99 with seq 0
Node #1  [  22.61617] Forward data with seq 0 via 2
Node #2  [  22.81623] Forward data with seq 0 via 13
Node #13 [  23.01630] Forward data with seq 0 via 14
Node #14 [  23.21637] Forward data with seq 0 via 24
Node #24 [  23.41644] Forward data with seq 0 via 25
Node #1  [  23.61617] Send data to 99 with seq 1
Node #1  [  23.61617] Forward data with seq 1 via 2
Node #25 [  23.61650] Forward data with seq 0 via 35
Node #2  [  23.81623] Forward data with seq 1 via 13
Node #35 [  23.81656] Forward data with seq 0 via 45
Node #13 [  24.01630] Forward data with seq 1 via 14
Node #45 [  24.01662] Forward data with seq 0 via 55
Node #14 [  24.21637] Forward data with seq 1 via 24
Node #55 [  24.21666] Forward data with seq 0 via 65
Node #24 [  24.41644] Forward data with seq 1 via 25
Node #65 [  24.41672] Forward data with seq 0 via 76
Node #1  [  24.61617] Send data to 99 with seq 2
Node #1  [  24.61617] Forward data with seq 2 via 2
Node #25 [  24.61650] Forward data with seq 1 via 35
Node #76 [  24.61679] Forward data with seq 0 via 85
Node #2  [  24.81623] Forward data with seq 2 via 13
Node #35 [  24.81656] Forward data with seq 1 via 45
Node #85 [  24.81686] Forward data with seq 0 via 86
Node #13 [  25.01630] Forward data with seq 2 via 14
Node #45 [  25.01662] Forward data with seq 1 via 55
Node #86 [  25.01691] Forward data with seq 0 via 96
Node #14 [  25.21637] Forward data with seq 2 via 24
Node #55 [  25.21666] Forward data with seq 1 via 65
Node #96 [  25.21697] Forward data with seq 0 via 97
```

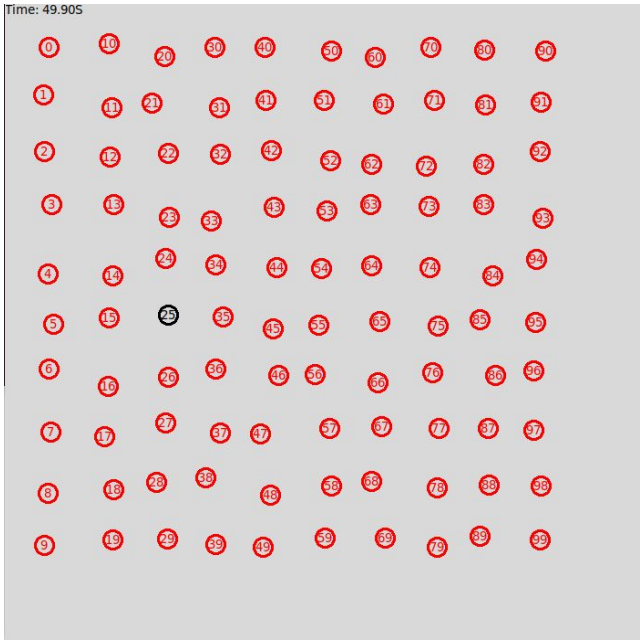
```
Node #24 [  25.41644] Forward data with seq 2 via 25
Node #65 [  25.41672] Forward data with seq 1 via 76
Node #97 [  25.41703] Forward data with seq 0 via 88
Node #1  [  25.61617] Send data to 99 with seq 3
Node #1  [  25.61617] Forward data with seq 3 via 2
Node #25 [  25.61650] Forward data with seq 2 via 35
Node #76 [  25.61679] Forward data with seq 1 via 85
Node #88 [  25.61710] Forward data with seq 0 via 89
Node #2  [  25.81623] Forward data with seq 3 via 13
Node #35 [  25.81656] Forward data with seq 2 via 45
Node #85 [  25.81686] Forward data with seq 1 via 86
Node #89 [  25.81718] Forward data with seq 0 via 99
Node #99 [  25.81723] Got data from 1 with seq 0
Node #13 [  26.01630] Forward data with seq 3 via 14
Node #45 [  26.01662] Forward data with seq 2 via 55
Node #86 [  26.01691] Forward data with seq 1 via 96
Node #14 [  26.21637] Forward data with seq 3 via 24
Node #55 [  26.21666] Forward data with seq 2 via 65
Node #96 [  26.21697] Forward data with seq 1 via 97
Node #24 [  26.41644] Forward data with seq 3 via 25
Node #65 [  26.41672] Forward data with seq 2 via 76
Node #97 [  26.41703] Forward data with seq 1 via 88
Node #1  [  26.61617] Send data to 99 with seq 4
Node #1  [  26.61617] Forward data with seq 4 via 2
Node #25 [  26.61650] Forward data with seq 3 via 35
Node #76 [  26.61679] Forward data with seq 2 via 85
Node #88 [  26.61710] Forward data with seq 1 via 89
Node #2  [  26.81623] Forward data with seq 4 via 13
Node #35 [  26.81656] Forward data with seq 3 via 45
Node #85 [  26.81686] Forward data with seq 2 via 86
Node #89 [  26.81718] Forward data with seq 1 via 99
Node #99 [  26.81723] Got data from 1 with seq 1
```



Слики број 2.1, 2.2, 2.3 - Screenshots од GUI на симулацијата AODV

Промена на симулацијата за Flood example-от

При изработката на проектната задача направени се следните модификации на изворниот код за flood example-от: при генерирањето на јазлите во мрежата со помош на рамномерната распределба сменети се нивните позициите (px, py) на интервалниот опсег [-10, 10], зголемена е вредноста на tx_range-от за broadcast на 150, сменето е id-то на source јазелот(SOURCE = 25) како за илустрација и намалено е времетрањето на целата симулација на 50 секунди. После направените промени на параметрите, извлечени се неколку позначајни заклучоци. Јазлите се збиени и поретко расфрлани едни од други со промената на координатите (px, py). Повеќе соседни јазли се опфатени/посетени преку единечен broadcast од даден јазел со промената на tx_range параметарот со вредност еднаква на 150. Најважно, пред се сите јазли се достапни (reachable) во мрежата пред истекот на симулацијата, односно за сите јазли recv property-то на класата MyNode е сетиран на True. Времетрањето на целата симулација е најмалку 10-тина секунди вклучително со времето за timeout, што значи и првичните 50 секунди за времетрањето на целата симулација се премногу.



Слика број 3 - Screenshots од GUI на симулацијата flood после промената на параметрите