به نام خداوند بخشنده مهربان

موضوع :

گزارش پروژه سوم شبکه
شبکه ی p2p

ارائه شده توسط:

دکتر خونساری

گردآوری شده توسط :

علیرضا زارع نژاد اشکذری

در:

دانشکده مهندسی برق و کامپیوتر دانشگاه تهران

ابتدا سعی می کنیم از روی فایل کانفیگ اطلاعات مورد نیاز را برای شبکه استخراج کنیم. بدین منظور مطابق توضیحات پروژه فایل config.txt را به صورت زیر آماده می کنیم.

```
6    // number of nodes in network
3    // number of maximum neighbor for each host
2    // each host sends hello message per seconds
8    // treshold time that each host should receive message from neighbors
10   // seconds that should we choose random host to be turned off
20   // each host should tured off for this period
5    // running program for this period
127.0.0.1, 4001, 1     // ip, port, id for each host
2 ,4002 ,127.0.0.2
3 ,4003 ,127.0.0.3
4 ,4004 ,127.0.0.4
5 ,4005 ,127.0.0.5
6 ,4006 ,127.0.0.6
```

در خط اول تعداد نود های شبکه ، در خط دوم ماکسیمم تعداد همسایه هایی که هر نود می تواند داشته باشد ، در خط سوم یه عدد که نشان دهنده تعداد ثانیه هایی است که هر هاست باید پیام ارسال کند ، در خط چهارم یک عدد می آید که نشان می دهد در صورتی که در این مدت اگر پیامی از همسایه ای دریافت نشود آن همسایه از همسایه های خود حذف می شود ، در خط بعدی عددی که می آید نشان می دهد که به صورت رندوم هر چند ثانیه یک بار رندوم یکی از هاست ها خاموش می شود و در خط بعدی مدت زمان خاموشی را نشان می دهد. در خط بعدی مدت زمان اجرای شبیه سازی بر حسب دقیقه و در خط بعدی نرخ اتلاف بسته ها آورده می شود. در خطوط بعدی مشخصات هر هاست آورده می شود.
پس از نوشتن فایل کانفیگ به صورت بالا کافی است تابعی داشته باشیم تا اطلاعات رو از فایل استخراج کرده و مقدار دهی اولیه نماید.

```
def read_config_file(file_path):
    global number_of_hosts, number_of_maximum_neighbors, hosts_send_hello_message_per_second
    global seconds_hosts_should_turned_off, period_off_second_for_host
    global minutes_of_running_program, hosts_info , treshold_second_for_being_neighbor, drop_rate

    with open(file_path) as file_pointer:
        for cnt, line in enumerate(file_pointer):
            if cnt == 0:
                number_of_hosts = int(line)
            elif cnt == 1:
                number_of_maximum_neighbors = int(line)
            elif cnt == 2:
                hosts_send_hello_message_per_second = int(line)
            elif cnt == 3:
                treshold_second_for_being_neighbor = int(line)
            elif cnt == 4:
                seconds_hosts_should_turned_off = int(line)
            elif cnt == 5:
                period_off_second_for_host = int(line)
            elif cnt == 6:
                minutes_of_running_program = int(line)
            elif cnt == 7:
                drop_rate = float(line)
            else:
                host_ip, host_port, host_id = line.strip("\n").split(", ")
                hosts_info.append([int(host_id), host_ip, int(host_port)])

```

در اینجا hosts_info لیستی از tuple ها برای هر هاست خواهد بود که شامل آیدی ، آدرس ip و پورت آن خواهد بود.

حال دو کلاس message , host را پیاده می کنیم که در ادامه توضیح داده می شود.

```
class Message:
    def __init__(self, sender_id, sender_ip, sender_port, message_type = "hello"
    , bidirectional_neighbors = []
    , unidirectional_neighbors = []
    , last_message_from_reciever_to_sender = 0
    , last_message_from_sender_to_reciever = 0):
        self.sender_id = sender_id
        self.sender_ip = sender_ip
        self.sender_port = sender_port
        self.bidirectional_neighbors = bidirectional_neighbors
        self.unidirectional_neighbors = unidirectional_neighbors
        self.last_message_from_reciever_to_sender = last_message_from_reciever_to_sender
        self.last_message_from_sender_to_reciever = last_message_from_sender_to_reciever

```

همان طور که در صورت پروژه فرض شده است، هر هاست هر دو ثانیه یک بار به همسایه های خود یک پیام hello ارسال می کند. محتوای این پیام شامل آیدی و آدرس ip و پورت فرستنده و همسایه های bidirectional و unidirectional خواهد بود.همچنین نوع بسته ی hello نیز صریحا در پیام ارسالی ذخیره می شود. همچنین زمانی که پیام از فرستنده به گیرنده و از گیرنده به فرستنده ارسال می شود را نیز در این پیام ارسال می کنیم.
همان طور که اشاره شد برای هاست ها یک class host ساخته ایم که یک متد به نام run دارد . برای هر هاست یه ترد به آن اختصاص می دهیم و تابع run آن را اجرا می کنیم.

همچنین یه ترد برای انتخاب کردن رندوم یک هاست برای خاموش شدن و یک ترد نیز برای terminate کردن برنامه استفاده می کنیم.

```
def start_network():
    list_of_hosts = []
    for i in range(number_of_hosts):
        list_of_hosts.append(Host(hosts_info[i][0], hosts_info[i][1], hosts_info[i][2], hosts_info))
        t = threading.Thread(target=list_of_hosts[i].run)
        t.start()
    t1 = threading.Thread(target=disable_random_host, args=(list_of_hosts,))
    t2 = threading.Thread(target=program_terminator)
    t1.start()
    t2.start()
```

مطابق تابع بالا ترد اول مخصوص انتخاب رندوم هاست جهت خاموش شدن هست که به عنوان ورودی لیست همه ی نود ها را دریافت می کند.
ترد دوم نیز برای اتمام برنامه استفاده می شود.

```
def disable_random_host(list_of_hosts):
    global terminate_program
    while True:
        num = np.random.randint(len(number_of_hosts))
        if terminate_program == True:
            break
        list_of_hosts[num].disable()
        time.sleep(seconds_hosts_should_turned_off)
```

برای خاموش کردن هاست ، از یک عدد رندوم استفاده می کنیم و تابع disable را برای هاست مورد نظر صدا می کنیم. در اینجا sleep استفاده شده است که مثلا پیش فرض پروژه ۱۰ ثانیه است ، بدین معنی که هر ۱۰ ثانیه یک هاست انتخاب شده و خاموش می شود.

```
def program_terminator():
    global terminate_program
    print("running network ...")
    for i in range(minutes_of_running_program):
        time.sleep(60)
        print("+1 minutes passed...")
    terminate_program = True
```

اتمام شبیه سازی نیز با ترد فوق انجام می شود که مثلا پیش فرض پروژه در پنج دقیقه اجرا می شود. برای این منظور از یک متغیر گلوبال به نام terminate program استفاده کردیم.که هر وقت توسط ترد فوق true شود سایر ترد های هر هاست که در ادامه ذکر می شود متوقف می شوند.

حال تابع disable کلاس هاست را بررسی می کنیم.

```
def disable(self):
    self.is_off = True
    t = threading.Thread(target=self.wait_to_enable)
    print("host with %s:%d with id %d turned off"%(self.host_ip, self.host_port, self.host_id))
    t.start()

def wait_to_enable(self):
    time.sleep(period_off_second_for_host)
    print("host with %s:%d with id %d turned on"%(self.host_ip, self.host_port, self.host_id))
    self.is_off = False
```

هر هاست یه متغیر دارد که مشخص میکند که هاست خاموش هست یا نه و با یک متغیر boolean آن را نمایش می دهیم. ابتدا آن را true کرده و سپس یک ترد اختصاص می دهیم و پس از sleep برای زمان مورد نیاز که پیش فرض پروژه ۲۰ ثانیه هست آن را false می کنیم.بدین منظور پس از sleep به مدت لازم که توسط کانفیگ به شبکه داده می شود هاست دوباره روشن می شود.
لازم به ذکر است هنگام خاموشی هر هاست اینطور فرض شده است که لیست همسایه های دو طرفه و یک طرفه clear می شوند و این در ادامه ی کد ها آورده می شود.
حال برای اطمینان از صحت پیاده سازی این بخش از پروژه آن را اجرا کرده و خروجی را در ترمینال مشاهده می کنیم. برای این منظور از print استفاده می کنیم.

```
  File "/usr/lib/python3.6/threading.py", line 916, in _bootstrap_inner
alireza@alireza-X550VXK:~/network$ /usr/bin/python3 /home/alireza/network/main.py
host with 127.0.0.5:4005 with id 5 turned off at 13:23:18.338670
running network ...
host with 127.0.0.2:4002 with id 2 turned off at 13:23:28.352009
host with 127.0.0.3:4003 with id 3 turned off at 13:23:38.359903
host with 127.0.0.5:4005 with id 5 turned on at 13:23:38.364525
host with 127.0.0.5:4005 with id 5 turned off at 13:23:48.367911
host with 127.0.0.2:4002 with id 2 turned on at 13:23:48.371754
host with 127.0.0.3:4003 with id 3 turned off at 13:23:58.375902
host with 127.0.0.3:4003 with id 3 turned on at 13:23:58.379724
host with 127.0.0.6:4006 with id 6 turned off at 13:24:08.383907
host with 127.0.0.5:4005 with id 5 turned on at 13:24:08.387718
host with 127.0.0.5:4005 with id 5 turned off at 13:24:18.391777
host with 127.0.0.3:4003 with id 3 turned on at 13:24:18.395699
+1 minutes passed...
host with 127.0.0.3:4003 with id 3 turned off at 13:24:28.400138
host with 127.0.0.6:4006 with id 6 turned on at 13:24:28.403715
host with 127.0.0.3:4003 with id 3 turned off at 13:24:38.407930
host with 127.0.0.5:4005 with id 5 turned on at 13:24:38.411740
host with 127.0.0.2:4002 with id 2 turned off at 13:24:48.417677
host with 127.0.0.3:4003 with id 3 turned on at 13:24:48.419736
host with 127.0.0.3:4003 with id 3 turned on at 13:24:58.427716
host with 127.0.0.1:4001 with id 1 turned off at 13:24:58.427991
host with 127.0.0.1:4001 with id 1 turned off at 13:25:08.435917
host with 127.0.0.2:4002 with id 2 turned on at 13:25:08.435997
host with 127.0.0.3:4003 with id 3 turned off at 13:25:18.443894
host with 127.0.0.1:4001 with id 1 turned on at 13:25:18.447735
+1 minutes passed...
```

همان طور که میبینیم هر ده ثانیه یک بار یک هاست رندوم انتخاب شده و به مدت ۲۰ ثانیه خاموش می باشد. و این در دو دقیقه اجرای آزمایشی قابل مشاهده است.

در ادامه کلاس هاست و توابع پیاده سازی شده آورده می شود. همان طور که دیده می شود هر هاست یک id , ip , port دارد و همچنین لیستی از همه ی هاست های موجود را دریافت می کند و یک متغیر دارد که نشان می دهد هاست مذکور روشن است یا نه و به وضوح یک متغیر برای ایجاد سوکت udp دارد و لیست همسایه های دو طرفه و یک طرفه را نگه می دارد. جهت ارائه گزارش باید تعدادی دیکشنری نگه داریم که مشخص سازد برای هر هاست آیدی چه تعداد پکت ارسال شده و یا از آن دریافت گردیده است و یا برای ارزیابی availability مدت زمان کانکشن را برای همسایه های دو طرفه نگه می داریم. همچنین آخرین زمان ارسال و دریافت پیام را نیز را برای هر هاست نگه می داریم.
دیکشنری توپولوژی در حقیقت برای اینکه برای هر هاست بدانیم چه همسایه های دو طرفه و یک طرفه ای دارند استفاده می کنیم و هر بار پس از دریافت پیام از هر نود آن را بروز می کنیم.

```
class Host():
    def __init__(self, host_id, host_ip, host_port, all_hosts):
        self.host_id = host_id
        self.host_port = host_port
        self.host_ip = host_ip
        self.all_hosts = all_hosts
        self.is_off = False
        self.socket = None

        self.bidirectional_neighbors = []
        self.unidirectional_neighbors = []
        self.temp_neighbor = None

        self.num_send_to_node = {}
        self.num_recieve_from_node = {}
        self.connection_time = {}
        self.last_message_from_node_to_here = {}
        self.last_message_to_node_from_here = {}
        self.topology = {}
        for host in all_hosts:
            self.topology[host[0]] = {}
            self.num_send_to_node[host[0]] = 0
            self.num_recieve_from_node[host[0]] = 0
            self.connection_time[host[0]] = 0
            self.last_message_from_node_to_here[host[0]] = None
            self.last_message_to_node_from_here[host[0]] = None
```

حال به بررسی متد run هر هاست می کنیم. در اینجا برای ارسال و دریافت پیام از هر هاست یک ترد اختصاص می دهیم و پس از آن که ترد های ارسال و دریافت پیام متوقف شدند یعنی متغیر global terminate program مقدار true شد به نوشتن خروجی و گزارش در فایل json متناظر با هر هاست می کنیم و آن را در تابع get report پیاده سازی کردم.

```
def run(self):
    # print("host %s:%d with id %s start"%(self.host_id, self.host_port, self.host_id))
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.socket.bind((self.host_ip, self.host_port))
    t1 = threading.Thread(target=self.recieve)
    t2 = threading.Thread(target=self.send)
    t1.start()
    t2.start()
    while not terminate_program:
        pass
    self.report()
```

```
def report(self):
    self.topology[self.host_id]["unidirectional_neighbors"] = self.unidirectional_neighbors
    self.topology[self.host_id]["bidirectional_neighbors"] = self.bidirectional_neighbors
    report = {}
    report['current_neighbors'] = self.bidirectional_neighbors
    report['topology'] = self.topology
    neighbors_report = {}
    for host in self.all_hosts:
        if not host[0] == self.host_id:
            neighbors_report[str(host)] = {}
            neighbors_report[str(host)]["available time"] = self.connection_time[host[0]]
            neighbors_report[str(host)]["available time percent"] = 100* self.connection_time[host[0]] / float(300)
            neighbors_report[str(host)]["send to"] = self.num_send_to_node[host[0]]
            neighbors_report[str(host)]["revieve from"] = self.num_recieve_from_node[host[0]]
    report["neighbors_report"] = neighbors_report
    with open(str(self.host_id) + ".json", "w") as outfile:
        json.dump(report, outfile, indent = 4)
```

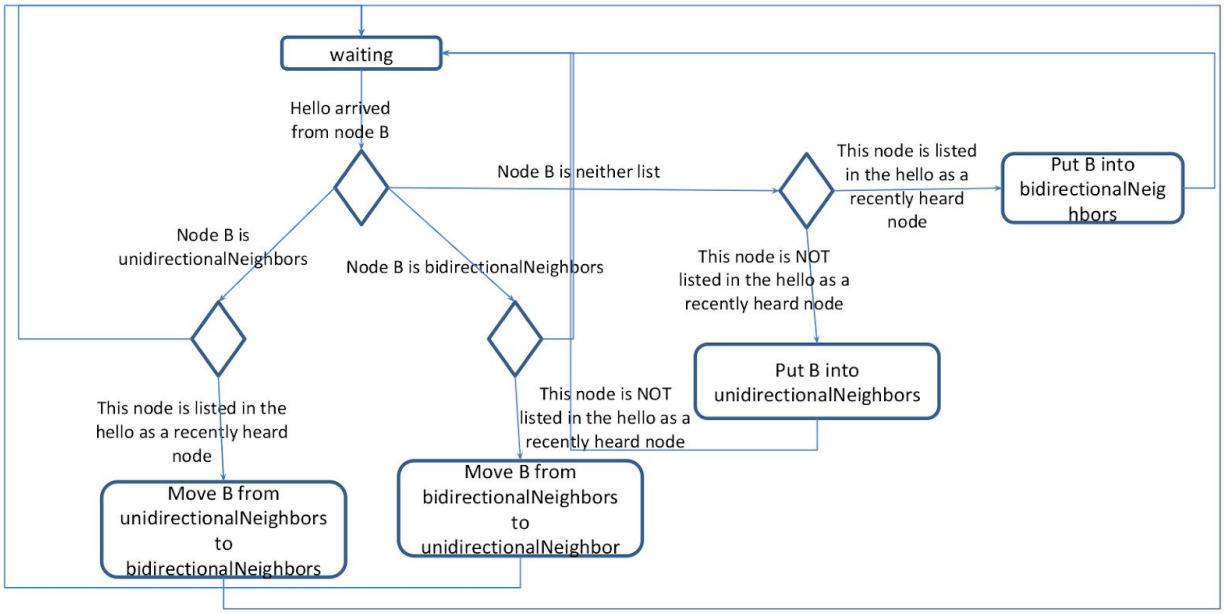
برای ارسال پیام نیز یک متد برای هاست در نظر گرفتیم که پیام message را مقدار دهی کرده و برای نود دریافتی با آدرس ip , پورت مشخص در سوکت ارسال می کند. توجه کنیم برای این کار از serialize کردن توسط تابع pickle.dumps انجام دادیم و در هنگام دریافت توسط گیرنده از load استفاده می کنیم تا کلاس مذکور دوباره ساخته شده و بتوانیم فیلد های مربوطه را استخراج کنیم.

```
def send_hello_message_to_node(self, host):
    now = time.time()
    message = Message(sender_id = self.host_id, sender_ip = self.host_ip, sender_port = self.host_port
    , bidirectional_neighbors = self.bidirectional_neighbors
    , unidirectional_neighbors = self.unidirectional_neighbors
    , last_message_from_sender_to_reciever = now
    , last_message_from_reciever_to_sender = self.last_message_from_node_to_here[host[0]])
    data = pickle.dumps(message)
    self.send_data_with_loss(data, (host[1], host[2]))
    self.num_send_to_node[host[0]] += 1
    self.last_message_to_node_from_here[host[0]] = now
```

همان طور که در صورت پروژه ذکر شد ، بسته های udp با نرخ اتلاف پنج درصد ارسال می شوند برای این کار نیز می توانیم در فرستنده و یا گیرنده از یک تابع random استفاده کنیم و در صورتی که از عدد بین صفر و یک ما از پنج صدم بیشتر بود پیام دریافت یا ارسال شود و در غیر این صورت drop می شود.

```
def send_data_with_loss(self, data, addr):
    if random.uniform(0,1) >= drop_rate:
        self.socket.sendto(data, addr)
    else:
        pass
        # print("drop message from host %s:%d to host %s:%s"%(self.host_ip, self.host_port, addr[0], addr[1]))
```

خب تا به اینجا کلیت و ساختار اولیه شبکه توضیح داده شد. مهم ترین بخش پروژه منطق موجود برای ارسال و دریافت پیام است و تغییراتی که هر کدام از لیست های unidirectional , bidirectional می کنند.
پیش فرض پیاده سازی تصویر زیر است.



هر کدام از هاست ها باید حداکثر دارای یه تعدادی همسایه باشند که مثلا پیش فرض پروژه ۳ تا است و خب در صورتی که کمتر این مقدار همسایه داشته باشند باید به صورت رندوم از میان هاست ها یکی انتخاب کنیم و به او پیام دهیم. برای این منظور اگر لیست unidirectional ها عضوی داشت یکی رندوم از آنجا انتخاب می کنیم. در غیر این صورت از کل هاست ها یک کاندیدا انتخاب می کنیم.

```
def find_new_neighbor(self):
    self.temp_neighbor = None
    candidate_neighbors = []
    if len(self.unidirectional_neighbors):
        self.temp_neighbor = self.unidirectional_neighbors[randrange(len(self.unidirectional_neighbors))]
    else:
        candidate_neighbors = [x for x in self.all_hosts if x not in self.bidirectional_neighbors and x[0]!=self.host_id]
        self.temp_neighbor = candidate_neighbors[randrange(len(candidate_neighbors))]
```

تابع فوق روند پیاده سازی را نشان می دهد.

- sendHelloToNeighbors
- Hello must include all heard neighbors
  - bidirectionalNeighbors
  - unidirectionalNeighbors
  - Not tempNeighbor
- Hello must be sent to
  - bidirectionalNeighbors
  - unidirectionalNeighbors
  - AND tempNeighbor

مطابق توضیحات فوق پیغام hello را به تمام همسایه های bidirectional , unidirectional می فرستیم. همان طور که قبلا توضیح داده شد این پیام شامل همسایه های یک طرفه و دو طرفه خواهد بود.

```
def send_to_all_neighbors(self):
    ready_to_send = []
    for node in self.bidirectional_neighbors:
        ready_to_send.append(node)
    for node in self.unidirectional_neighbors:
        ready_to_send.append(node)
    if self.temp_neighbor is not None:
        ready_to_send.append(self.temp_neighbor)

    for node in ready_to_send:
        self.send_hello_message_to_node(node)
        ready_to_send.remove(node)
```

اگر همسایه ها تعدادشون کم باشد، آن هاست رندوم را در temp neighbor قرار می دهیم و بسته های هلو را برای آن ، همچنین همسایه های یک طرفه و دو طرفه ارسال می کنیم.
در نهایت تابع send به شکل مقابل در خواهد آمد.

```
def send(self):
    while True:
        if terminate_program:
            break
        if not self.is_off:
            # print("host %s:%d with id %d ready to send message at %s"%(self.host_ip, self.host_port,
            #  self.host_id, datetime.datetime.now().time()))
            if len(self.bidirectional_neighbors) < number_of_maximum_neighbors:
                self.find_new_neighbor()
            self.send_to_all_neighbors()


        else:
            # print("jost %s:%d with id %d is off now to send."%(self.host_ip, self.host_port, self.host_id))
            self.unidirectional_neighbors.clear()
            self.bidirectional_neighbors.clear()

        time.sleep(hosts_send_hello_message_per_second)
    return
```

خب همان طور که در کد دیده می شود یک WHILE داریم اگر برنامه متوقف شود ترد فوق از حلقه خارج می شود. اگر هاست خاموش باشد ارسال به همسایه ها با شرایط مذکور آورده می شود. در غیر این صورت لیست های نگه داشته شده ریست می شوند.
در انتها یک sleep گذاشته شده که مشخص کند که ارسال بسته ها هر چند ثانیه یک بار انجام می شود.

قبل از آنکه به توضیح تابع receive بپردازیم لازم است به تابع زیر اشاره شود.

```
def not_heard_for_long_time(self, host):
    time.sleep(treshold_second_for_being_neighbor)
    now = time.time()
    if now - self.last_message_from_node_to_here[host[0]] > treshold_second_for_being_neighbor:
        if host in self.bidirectional_neighbors:
            self.bidirectional_neighbors.remove(host)
            self.connection_time[host[0]] += now - self.last_message_from_node_to_here[host[0]]
        elif host in self.unidirectional_neighbors:
            self.unidirectional_neighbors.remove(host)
    return
```

در این تابع هر وقت همسایه جدیدی به ما اضافه شد چک می کنیم که هر هشت ثانیه یک بار آیا پیامی از او دریافت کرده ایم یا نه. اگر پیامی دریافت نکنیم اگر همسایه دوطرفه باشد ، آن را از لیست همسایه های دو طرفه خارج کرده و زمان اتصال آن را بروز می کنیم در غیر این صورت کافی است از لیست همسایه های یک طرفه خارج شود.

در مورد تابع receive ، هنگامی که پیامی دریافت می شود ، ابتدا چک می کنیم که sender در لیست همسایه های دو طرفه یا یک طرفه ما وجود دارد یا نه. این به معنی last heard بودن خواهد بود. اگر آن در نود در لیست همسایه های یک طرفه بود آن را از آنجا پاک می کنیم. در صورتی که تعداد همسایه های دو طرفه ما از ماکسیمم کمتر بود آن را به آنجا اضافه می کنیم و ترد چک کردن اینکه در هشت ثانیه پیام دهد را صدا می کنیم.

```python
def recieve(self):
    global terminate_program
    while True:
        if terminate_program:
            break
        if not self.is_off:
            data, addr = self.socket.recvfrom(12048)
            message = pickle.loads(data)
            now = time.time()
            self.num_recieve_from_node[message.sender_id] += 1
            sender_info = (message.sender_id, message.sender_ip, message.sender_port)

            self.topology[sender_info[0]]["bidirectional_neighbors"] = message.bidirectional_neighbors
            self.topology[sender_info[0]]["unidirectional_neighbors"] = message.unidirectional_neighbors

            if ((self.host_id, self.host_ip, self.host_port) in message.unidirectional_neighbors
            or (self.host_id, self.host_ip, self.host_port) in message.bidirectional_neighbors):
                if sender_info not in self.bidirectional_neighbors:
                    if sender_info in self.unidirectional_neighbors:
                        self.unidirectional_neighbors.remove(sender_info)
                    if len(self.bidirectional_neighbors) < number_of_maximum_neighbors:
                        self.bidirectional_neighbors.append(sender_info)
                        if self.connection_time[sender_info[0]] == None:
                            self.connection_time[sender_info[0]] = now
                        t = threading.Thread(target = self.not_heard_for_long_time, args=(sender_info,))
                        self.last_message_from_node_to_here[message.sender_id] = now
                        t.start()
```

حال اگر last heard نبود، اگر جز همسایه های دو طرفه بود آن را از آنجا پاک می کنیم. در نهایت به همسایه uni خود اضافه می کنیم.
زمان دریافت پیام از آن را نیز باید بروز کرد.

کد این قسمت بخش receive نیز در زیر آورده شده است.

```python
    else:
        if sender_info not in self.unidirectional_neighbors:
            if sender_info in self.bidirectional_neighbors:
                self.bidirectional_neighbors.remove(sender_info)
                self.connection_time[message.sender_id] += now - self.last_message_from_node_to_here[message.sender
            self.unidirectional_neighbors.append(sender_info)
            self.last_message_from_node_to_here[message.sender_id] = now
            t = threading.Thread(target = self.not_heard_for_long_time, args=(sender_info,))
            t.start()
else:
    self.unidirectional_neighbors.clear()
    self.bidirectional_neighbors.clear()
```

پس از اجرای کد خروجی ها در فایل هایی با آیدی هر هاشت شامل اطلاعاتی خواسته شده قرار می گیرد.