Eötvös Loránd Tudományegyetem

Informatikai Kar
Informatikatudományi Intézet

Információs Rendszerek Tanszék

Az OSPF forgalomirányítási protokoll megvalósítása Python nyelven

Szerző:

Ambrus Lili Emma

Programtervező informatikus BSc.

Témavezető:

Kecskeméti Károly

PhD hallgató

Budapest, 2025

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:
Név: Ambrus Lili Emma
Neptun kód: V4ASUI

Képzési adatok:
Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)
Tagozat : Nappali
Belső témavezetővel rendelkezem

Témavezető neve: Kecskeméti Károly
munkahelyének neve, tanszéke: **ELTE IK, Információs rendszerek Tanszék**
munkahelyének címe: **1117, Budapest, Pázmány Péter sétány 1/C.**
beosztás és iskolai végzettsége: **PhD hallgató, MSc**

A szakdolgozat címe: Az OSPF forgalomirányítási protokoll megvalósítása Python nyelven

A szakdolgozat témája:
(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

A szakdolgozat célja az OSPF (Open Shortest Path First) széles körben használt forgalomirányítási protokoll egy egyszerűsített, demonstrációs és oktatási célokra alkalmas változatának elkészítése.
Az elkészítendő szoftver implementálni fogja a protokoll főbb funkcióit. Ezek közt, a teljes igénye nélkül szerepel a szomszédsági kapcsolatok kiépítése, a kapcsolat-állapot adatbázis felépítése és a legrövidebb útvonalak azonosítása.
A szoftver tartalmazni fog több a protokoll működésének megértését elősegítő elemet, mint például a hálózati gráf aktuális állapotának vizualizációját és a protokoll működése során generált hálózati csomagok naplózását.
A célkitűzések közt szerepel, hogy az elkészült munkát virtuális környezetben különböző hálózati forgatókönyvek segítségével részleteiben teszteljük, illetve kiértékeljük.
A projekt mélyebb betekintést nyújt a kapcsolat-állapot alapú forgalomirányítási protokollok működésébe és azok szerepébe a modern IP-hálózatokban.
A fejlesztés Python nyelven fog megvalósulni.

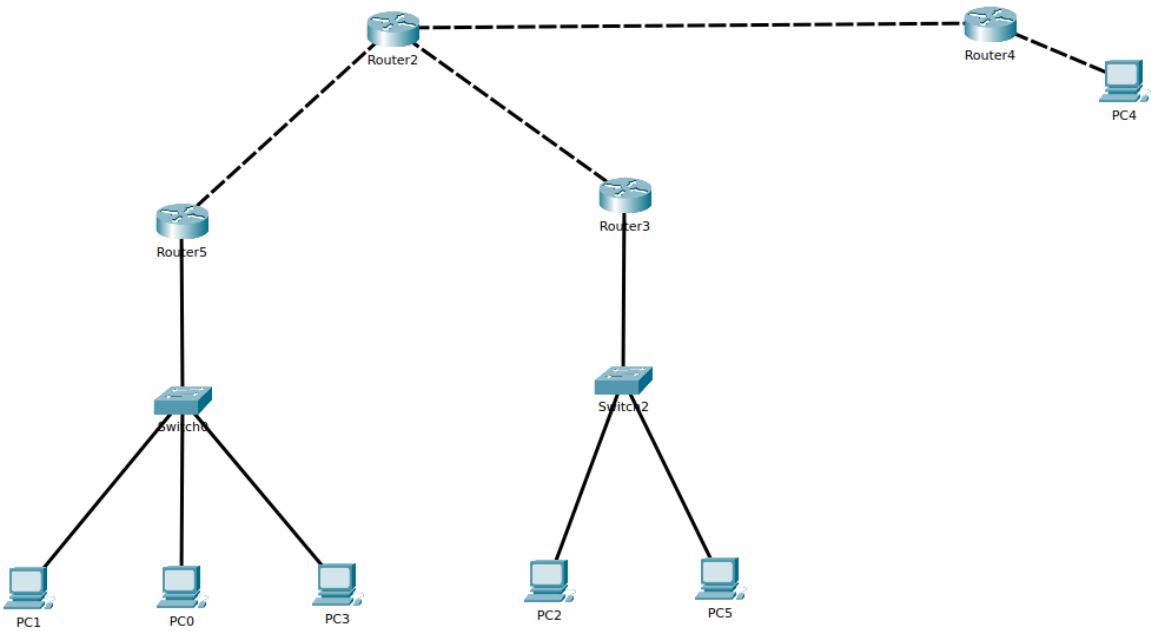
Budapest, 2024. 10. 15.

Tartalomjegyzék

1. Bevezetés

A modern számítógépes hálózatok megbízható működésének alapfeltétele a hatékony útvonalválasztás. A forgalom irányítása különösen nagy, összetett rendszerekben válik kulcsfontosságúvá, ahol a topológia folyamatosan változhat, és gyorsan kell megtalálni a legrövidebb vagy legbiztonságosabb útvonalat. Ebben a környezetben nyer különös jelentőséget az OSPF (*Open Shortest Path First*) protokoll, amelyet világszerte számos hálózat alkalmaz.

Az OSPF egy széles körben használt kapcsolat-állapot alapú forgalomirányítási protokoll, mely a legrövidebb útvonalak kiszámításával biztosítja a hálózati forgalom hatékony irányítását. Az *Internet Engineering Task Force* (IETF) fejlesztette azzal a céllal, hogy nagy autonóm rendszereken belül a hálózati csomagokat hatékonyan mozgassa. Az OSPF protokoll jelentősége abban rejlik, hogy képes kezelni a nagy és összetett hálózatokat, gyors konvergenciát biztosít, és támogatja a több útvonal egyidejű használatát, növelve ezzel a hálózat redundanciáját és megbízhatóságát.



1. ábra: Hálózati topológia

1.1. Szakdolgozat célja

A témaválasztásomat nemcsak a Python nyelvű fejlesztési lehetőség motiválta, hanem az is, hogy szerettem volna jobban megérteni egy ilyen típusú protokoll működését – nem csupán elméleti, hanem gyakorlati szempontból is. Erre a célra tökéletes alapot nyújtott az OSPF, mivel egy jól dokumentált és szabványosított protokollról van szó. A dolgozatom során az OSPFv2 (RFC 2328) protokoll egy leegyszerűsített változatának implementálását tűztem ki célul.

A szakdolgozat célja egy egyszerűsített, demonstrációs és oktatási célokra alkalmas OSPF változat elkészítése Mininet környezetben. Az elkészítendő szoftver implementálni fogja a protokoll főbb funkcióit, beleértve a szomszédsági kapcsolatok kiépítését, a kapcsolat-állapot adatbázis felépítését és a legrövidebb útvonalak azonosítását.

A dolgozat további részei részletesen bemutatják a fejlesztett szoftver felhasználói és technikai oldalát, valamint a tesztelés során kapott eredményeket. A fejlesztés Python nyelven történt és megvalósítás során a Mininet hálózatszimulációs környezetet használtam, amely lehetővé tette a különböző hálózati helyzetek kipróbálását és elemzését.

2. Felhasználói dokumentáció

2.1. A megoldott probléma rövid megfogalmazása

A szoftver implementálja az OSPF útirányítási protokoll főbb funkcióit, beleértve a szomszédsági kapcsolatok kiépítését, a kapcsolat-állapot adatbázis felépítését és a legrövidebb útvonalak azonosítását. Ezekhez mind különböző hálózati csomagokat használ az információ átadásához.

2.2. A felhasznált módszerek

A projekt során a következő módszereket és eszközöket használtam:

- **Mininet:** A Mininet egy hálózati szimulációs eszköz, amely lehetővé teszi virtuális hálózatok gyors és hatékony létrehozását és tesztelését. A Mininet segítségével különböző hálózati topológiákat és forgatókönyveket szimuláltunk, hogy teszteljük az OSPF protokoll működését és teljesítményét.
- **Wireshark:** A Wireshark egy hálózati forgalom elemző eszköz, amely lehetővé teszi a hálózati csomagok részletes vizsgálatát és naplózását. A Wireshark segítségével elemeztük az OSPF protokoll által generált hálózati csomagokat, hogy megértsük a protokoll működését és az adatcsomagok közötti kommunikációt.
- **Python:** A szoftver fejlesztése Python nyelven történt, amely lehetőséget biztosít a gyors és hatékony prototípus-készítésre, valamint a könnyen érthető és karbantartható kód írására.
- **Python könyvtárak:** A Mininet keretein belül a hálózati kommunikáció ponstosabb reprezentációja érdekében különböző könyvtárakat használtam, köztük a Scapy-t, a hálózati csomagok szimulálására, amivel a csomag rétegeit felhasználóbarát módon jeleníthetem meg. Emellett fontos szerepet játszott a NetworkX, a legrövidebb út kiszámolásában és a topológia felépítésében és reprezetnálásában.

2.3. A program használatához szükséges összes információ

A futtatáshoz használt rendszer felépítése és a program futtatása nem éppen intuitív, ezért a következőkben szeretném ennek a folyamatát minél közérthetőbben leírni. A virtuális környezet felépítésére különböző módszerekkel van lehetőség, én ezek közül kettőt szeretnék bemutatni. Az egyik a Virtualbox letöltését és konfigurálását igényli a második, pedig azzal az eshetőséggel számol, hogy a felhasználó Linux operciós rendszeren, natívan szeretné futtatni, mind a Mininetet, mind pedig a programot.

2.3.1. Rendszerkövetelmény

2.3.1.1. Virtualbox-al való futtatás

2.3.1.2. Lokális futtatás

2.3.2. Telepítési lépések Mininet telepítése:

Töltse le és telepítse a Mininet virtuális gépet a Mininet hivatalos oldaláról. Importálja a Mininet VM-et a VirtualBox vagy más VM programba, és indítsa el.

Egyéb szükséges beállitások:

- Settings -> System -> Processor -> Enable PAE/NX
- Settings -> Network -> Adapter 1 -> Attached To -> Bridged Adapter

Ha Linux operacios rendszert hasznal, akkor van lehetoseg a Mininet helyi letoltesere es a szimulacio nativ futtatasara.

2.3.3. Wireshark telepítése

Töltse le és telepítse a Wireshark programot a Wireshark hivatalos oldaláról.

2.3.4. Python és szükséges csomagok telepítése

Nyisson egy terminált és ssh segítségével csatlakozzon fel a MininetVM-re:

ssh mininet@[IP cím amin a MininetVM fut]

Telepítse a Python legfrissebb verzióját a Python hivatalos oldaláról. (Ez csak akkor szukseges, ha Linux operacios rendszeren helyileg lett telepitve a Mininet) Telepítse a szükséges Python csomagokat a requirements.txt fájl segítségével:

```
sudo pip install -r requirements.txt
```

2.3.5. A program futtatása

A korábban nyitott és felcsatlakoztatott terminált tovább használjul. Először is a OSPF kódot be kell vinni a Mininet virtuális gépbe:

```
scp -r Downloads/thesis mininet@[ IP cím amin fut a MininetVM ]:~/thesis
```

OSPF szimuláció futtatása:

```
sudo python3 run.py [ man | auto | test ]
```

Van lehetőség a szimuláció automatikus futtatására. Ehhez nyissa meg a Mininet terminálokat az egyes routerekhez:

```
xterm [ routerek neve ]
```

Majd futtassa az OSPF szimulációt az egyes routerek terminálján:

```
sudo python3 ospf.py [ router neve ]
```

Wireshark használata:

Töltse le a Mininet virtuális gépből a logs mappában levő pcap hálózati csomagokat tartalmazó fájlokat:

```
scp -r mininet@[ IP cím amin a MininetVM fut ]:~/thesis ~/Downloads
```

Indítsa el a Wiresharkot, és nyissa meg a megfelelő fájlt:

Minden fájl a következő név formátummal található a logs mappában: [Router neve]_[Interfész neve].pcap

Elemezze az OSPF által generált hálózati csomagokat a Wireshark segítségével.

2.4. A program funkciói

- **Szomszédsági kapcsolatok kiépítése:** A program implementálja az OSPF Hello protokollt, amely lehetővé teszi a routerek számára, hogy felfedezzék és kapcsolatot létesítsenek a szomszédos routerekkel.
- **Kapcsolat-állapot adatbázis felépítése:** A program ebbe az adatbázisba összegyűjti és tárolja a hálózat topológiai adatait, amelyeket a routerek közötti kapcsolat-állapot hirdetések (LSA) segítségével frissít.
- **Legrövidebb útvonalak azonosítása:** A program a Dijkstra algoritmust használja a legrövidebb útvonalak kiszámítására, amely lehetővé teszi a routerek számára, hogy meghatározzák a legoptimálisabb útvonalakat a hálózaton belül.
- **Log-ok terminálba megjelenítése:** A routerek közötti kommunikáció olvashatósága érdekében az összes router minden hálózati kommunikációt és állapotváltozási döntést, lejegyez, egy, a nevével ellátott fájlba, és ezt egy Monitoring modul a felhasználónak megjeleníti.
- **Hálózati csomagok bejegyzése:** A routerek közötti hálózati csomagokkal történő kommunikációt, az egyéb bejegyzésektől külön tároljuk, majd a felhasználó a Wireshark segítségével ezeket elemezheti.

3. Fejlesztői dokumentáció

3.1. A program specifikációja

3.1.1. Használati esetek

2. ábra: Használati eset diagram

3.2. Felhasznált módszerek

A program tisztán Pythonban lett megírva, azonban az adattároláshoz még megjelennek mellette különböző fájlok.

3. ábra: A program nyelve

A virtuális hálózat és annak tagjainak konfigurációit egy yaml konfigurációs fájl tárolja. Az algoritmus által log-olt hálózati csomagokat router interfészenként külön fájlokra bontva pcap kiterjesztésű fájlok tárolják. Ezeknek hatékony olvasásához a feljebb bemutatott Wireshark hálózati elemző programot ajánlom.

A program objektum-orientált módon íródott, logika ás felhasználás szerint modulokra bontva. Emelle

3.3. Hasznos fogalmak

RID:

AreaID:

Hello Protocol:

Hello interval

Dead interval

3.4. A program felépítése

A programot logikai szempontból három fő modulra lehet szétbontani. A Hálózatkezelő modulra, ami a hálózati elemeket közvetlen manipulálja, OSPF modulra, ami a Routereken

futó OSPF algoritmust célozza szimulálni és a Monitoring modul, ami a routerek közötti kommunikációt elemzi és ez alapján épít kapcsolatokat, adatbázist és dönti el a topológiát.

Emellett megjelenik a common mappa, ami a több programrész által használt **get_config** yaml konfigurációs fájl parser-t tartalmazza. A tests mappa, ami két fő modul, a hálózatkezelő rendszer és az OSPF tesztelési elemeit és az ehhez szükséges konfigurációs fájlokat tartalmazza.

4. ábra: Komponensdiagram

3.4.1. Monitoring

3.4.2. Network

A Hálózatkezelő rendszer építi fel az alapot az algoritmus futtatásához. Létrehozza és elindítja a **Mininet** virtuális hálózati példányt, a **LogMonitor**-t, ami a fájlok folyamatos figyelésével jeleníti meg a routerek által generált új információt. Létrehozza azt a konfigurációs fájl alapján azt a **Topology** példányt, ami alapján a virtuális hálózat felépül, majd ezek után beállítja a routerek interfészeinek konfigurációját. Emellett ez a modul felel a routereken futó program elindításáért, majd leállásának kezeléséért.

5. ábra: Network osztálydiagramja

3.4.2.1. Class::NetworkManager

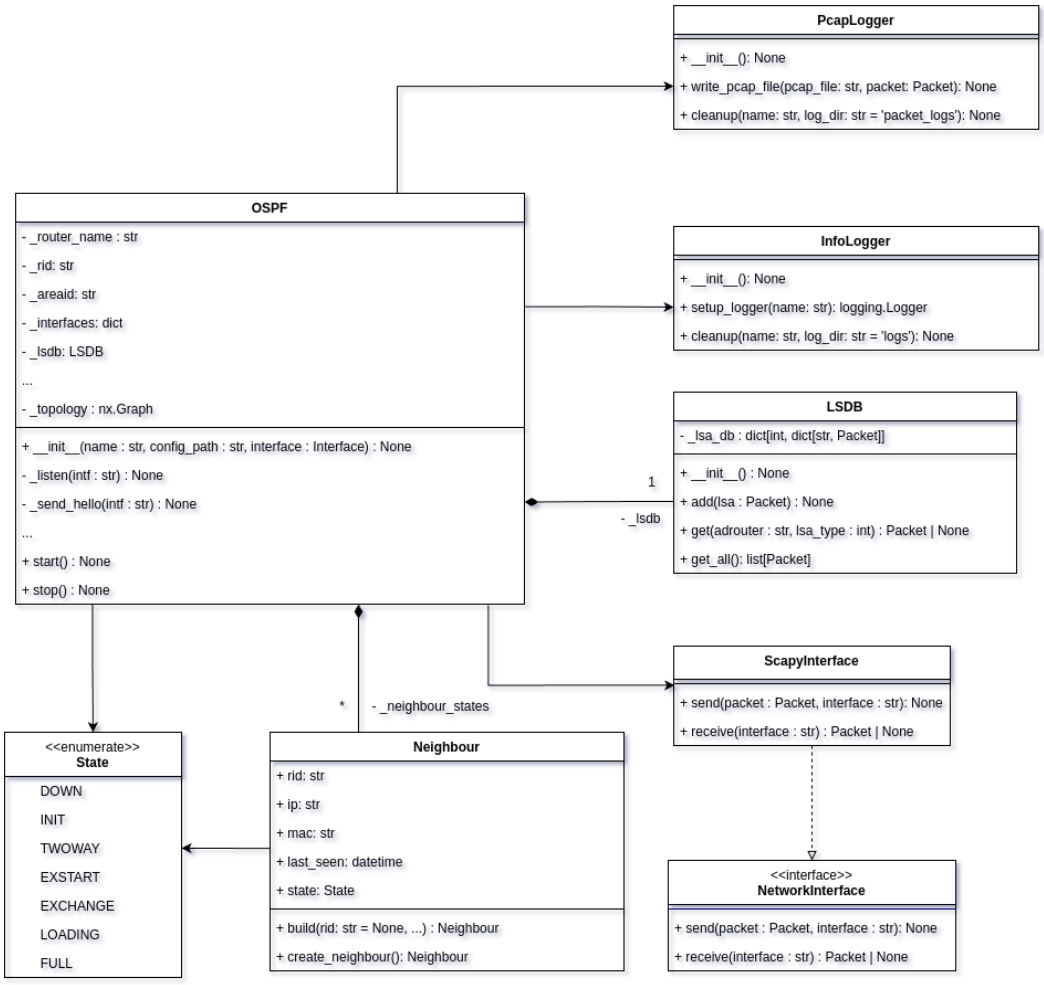
3.4.2.2. Class::LogMonitor

3.4.2.3. Class::Topology

A **Topology** osztály a Mininet **Topo** osztályát felülírva a konfigurációs fájl alapján felépíti. Létrehozza a hálózati eszközöket és a fájl alapján beállítja közöttük a kapcsolatokat.

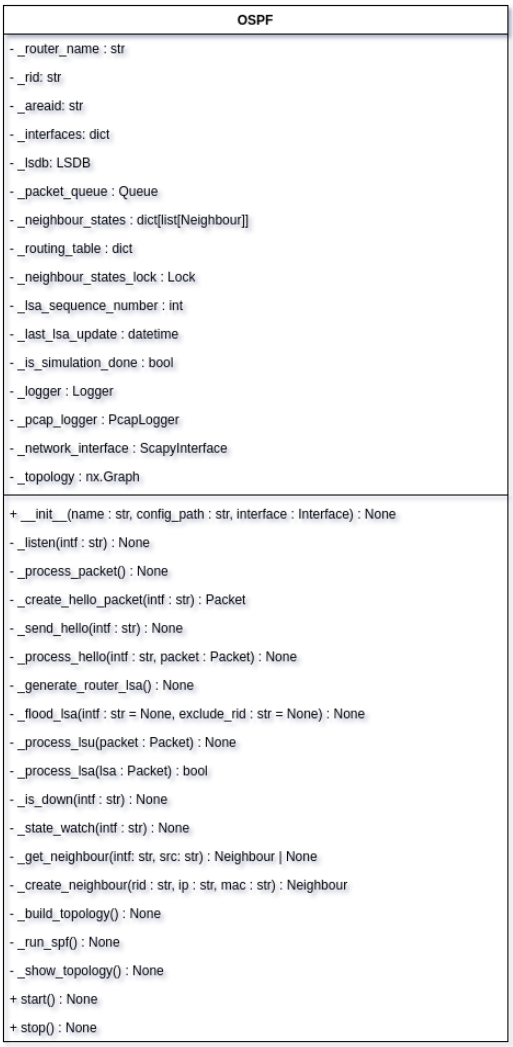
3.4.2.4. Class::ScapyInterface

3.4.3. OSPF Core



6. ábra: OSPF Core osztálydiagramja

3.4.3.1. Class::OSPF



7. ábra: OSPF osztálydiagramja

3.4.4. Log fájlok

Negyedik modulnak nem tekinthető, azonban az algoritmus felhasználóbarát megjelenéséhez és a mélyebb hálózati kommunikáció értelmezéshez elengedhetetlenek az OSPF modul által generált log fájlok. Ezeket felhasználási módszerek alapján két mappába osztottam szét.

3.4.4.1. Logs

A logs mappába az **InfoLogger** minden lényeges történést, állapotváltást, csomagküldést és fogadást és a végső topológiát a routerek szempontjból lement egy, a router nevével ellátott log fájlba és ezeket a **LogMonitor** kiírja a fő terminálra.

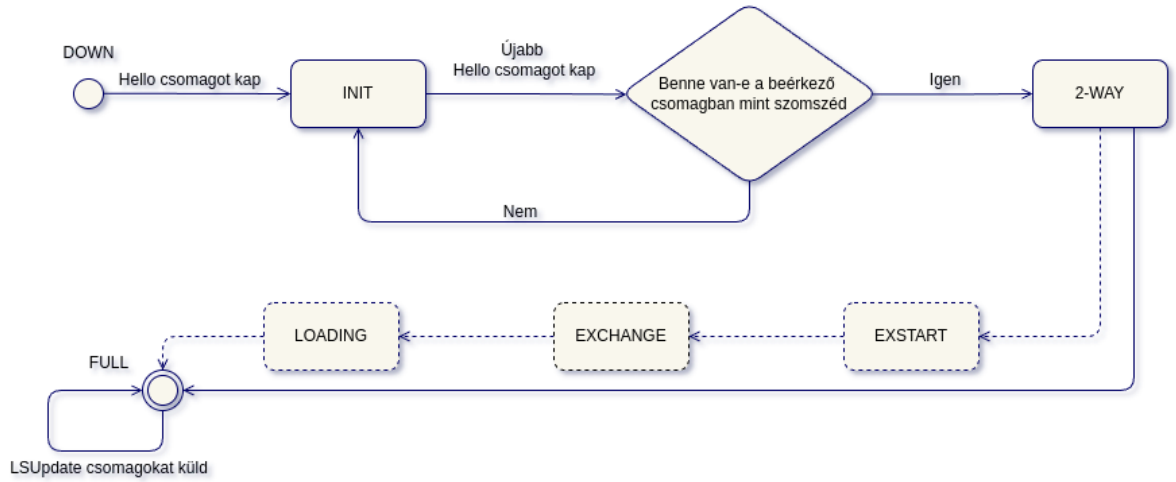
3.4.4.2. Packet_logs

A routerek hálózati interfészein küldött hálózati csomagokat a **PcapWriter** interfészenként minden küldés és fogadásnál elmenti pcap fájlokba. A pcap fájlok kimondottan a hálózati

csomagok tárolására alkalmas és amikor az algoritmus befejeződött letöltés után a Wireshark-ban megnyithatóak és részleteiben lehet elemezni, hogyan is néz ki egy OSPF hálózati csomag.

3.4.4.3. OSPF szomszéd állapotok

Az algoritmus lefutása során a routerek interfészei különböző szomszédossági kapcsolati állapotokon mennek keresztül a kommunikáció és a korábbi állapotok alapján. A kezdeti még nem működést reprezentáló állapottól a teljesen működőképes és szinkronizált állapotig. 10.1. Neighbor states



8 .ábra: OSPF állapotdiagram

Down: Kezdeti állapot, amely jelzi, hogy a szomszéddal még nem történt kommunikáció, vagy mostanában nem történt kommunikció és nem-működőképesnek véljük őt. Mivel a Hello Protocol, a meghatározott időközönként küldött Hello csomagok segítségével felügyeli a szomszédok elérhetőségét, ha egy router még nem küldött vagy már túl régen, a Dead interval-on kívül küldött csomagot, akkor DOWN állapotba helyezzük.

Init:

2-Way:

Full:

A **`2-WAY`** és a **`FULL`** állapotok közötti állapotokban; **`EXSTART`**, **`EXCHANGE`**, **`LOADING`** a szomszédossági kapcsolat formálása, a Master/Slave kapcsolat kiépítése és egy folyamatos flood-olás álatali adatbázis szinkronizáció történik, ezek jelenleg csak reprezentatív szereppel jelennek meg a kódban és tényleges implementációra várnak.

Jelenleg az adatbázis szinkronizációt a **`FULL`** állapotban kezeljük, a routerek szomszédokat új kapcsolatként kezelve LSUpdate hálózati csomagok segítségével terjeszti a szomszédokról szerzett és LSDB-jében tárolt információt a többi szomszédjának. Ezzel minden szomszéd ismerni fogja mindegyik szomszédjának a kapcsolatait és felépülhet a topológia.

3.5. Tesztelési terv

A teszteseteket két fő csoportra szedtem a modulok alapján, OSPF tesztek és Hálózati tesztek. A teszteket a pytest könyvtár használatával írtam és futtattam. A tesztfájlok a következőképpen futtathatók:

```
sudo pytest –v tests/
```

A tesztek sudo nélküli futtatása a Mininetet használó tesztek átlépését eredményezi, hiszen ezek futtatása igényli a rendszerszintű jogosultság megadását.

3.5.1. Tesztkörnyezet

Python verzió: Python 3.10

Mininet verzió: Mininet 2.3.0

Ubuntu verzió: Ubuntu 22.04 LTS

Python könyvtárak: A requirements.txt-ben felsoroltak.

3.5.2. OSPF egységtesztek

3.5.2.1. Teszteset: Létrejön az OSPF osztály

Bemenet	Az OSPF osztály konstruktora megkapja az 1.1.1.1-es RID-val rendelkező router konfigurációját.
Elvárt kimenet	Az osztály helyesen kiolvassa a konfigurációt.
Tényleges kimenet	Az OSPF osztály helyesen eltárolja a konfigurációt a teszteset konfigurációs fájljából.

3.5.2.2. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.5.2.3. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.5.2.4. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.5.2.5. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.5.3. Hálózat egységtesztek

3.5.3.1. Teszteset: Topológia felépülése a minimálisan szükséges konfigurációval

Bemenet	A Topology osztály kap egy, a szükséges paramétereket tartalmazó konfigurációs fájlt.
Elvárt kimenet	Létrejön a topológia.
Tényleges kimenet	A konfigurációs fájl alapján sikeresen létrejön a topológia.

3.5.3.2. Teszteset: Topológia felépülése több-eszközös konfigurációval

Bemenet	A Topology osztály kap egy több eszköz konfigurációját tartalmazó fájlt.
Elvárt kimenet	Az osztály értelmezi a konfigurációt és a kapcsolatokat és felépíti a topológiát.
Tényleges kimenet	Az osztály helyes értelmezés után felépíti a helyes topológiát.

3.5.3.3. Teszteset: Létrejön a NetworkManager és a Mininet virtuális hálózat is

Bemenet	Létrehozunk egy NetworkManager osztályt.
Elvárt kimenet	Beolvassa a konfigurációt és létrehozza a LogMonitor-t, a Topology-t és létrehozza a Mininet hálózatot is.
Tényleges kimenet	Beolvassa a router.yml konfigurációs fájlt és létrehozza a szükséges eszközöket.

3.5.3.4. Teszteset: A routerek interfészei helyesen konfiguráltak

Bemenet	Létrehozza a Topology-t és a Mininet hálózatot, majd elindítja a hálózatot.
Elvárt kimenet	Az R1 router eth0 interfésze a fájlnak megfelelően lett konfigurálva.
Tényleges kimenet	Az R1 router interfész neve és IP címe helyesen konfigurált.

3.5.3.5. Teszteset: Ha elindul a hálózat akkor el tud indulni az OSPF is

Bemenet	Elindítjuk a létrehozott Mininet hálózatot és elindítjuk a routereken az OSPF folyamatokat.
Elvárt kimenet	Ha megnézzük egy router folyamatait, megjelenik az ospf.py, mint futó folyamat.
Tényleges kimenet	Az ospf.py megjelenik a futó folyamatok között.

4. További fejlesztési lehetőségek

A program rendben bemutatja az OSPF útkeresési algoritmus hasznát és főbb funkcióit. De ez további fejlesztéssel még részletesebben bemutathatná a felhasználó számára milyen rejtett folyamatok futnak le mielőtt eldöntik a routerek hol szerepelnek a topológiában és merre irányítsák a kommunikációt.

Teljes OSPF-állapot implementáció: A program jelenleg az algoritmus működéséhez elengedhetetlen állapotokat implementálja, azonban ez továbbfejleszthető lenne, amivel mélyebb betekintést nyerhetünk a döntéshozatal indoklásába.

Több-területűség támogatása: Demonstrációs célból a program egy hálózati területen belüli működését implementálja. A valóságban azonban vannak több területű hálózatok is. Egy még realisztikusabb demonstráció érdekében ennek a támogatottságát is implementálni lehet.

Konfigurációsfájl validáció: A program jelenleg nem támogatja a felhasználó által megadott konfigurációkat, de a saját konfiguráció megadására a támogatottság létezik. Egy validációs modul hozzáadásával, a programnak nem kellene bíznia abban, hogy a felhasználó jól adja meg a konfigurációt az eszközökhöz, hanem rossz input esetén elutasítaná azt és iránymutatást adna a hiba kijavításában.

5. Irodalomjegyzék

[1] https://datatracker.ietf.org/doc/html/rfc2328

[2] https://mininet.org/api/index.html