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.

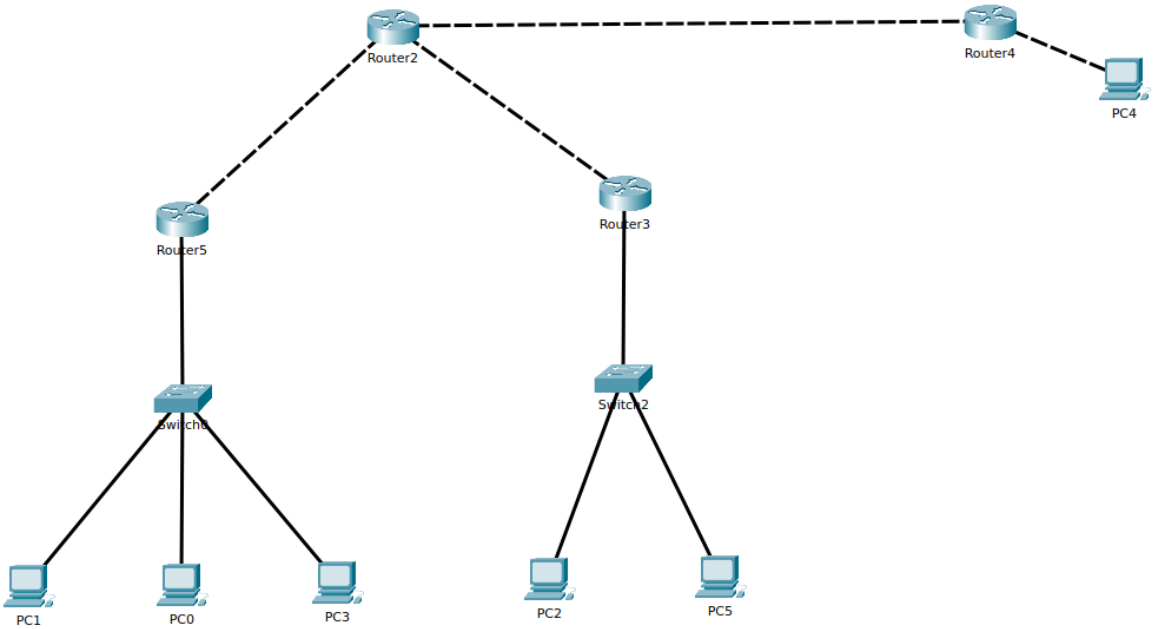
Budapest, 2024. 10. 15.

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. (RFC 2328 [1])



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 [1]) 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 egyszerűsített módon 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 program funkciói

A program implementálja a következő OSPF és naplózási funkciókat tartalmazza:

- **Szomszédsági kapcsolatok (Hello protokoll):** A rendszer a Hello protokoll segítségével építi ki a routerek közötti szomszédossági kapcsolatot. Minden router meghatározott időközökben Hello csomagot küld egy, OSPF kommunikációra lefoglalt multicast címre. Azok a routerekkel, akik válaszolnak neki egy saját Hello csomaggal, kölcsönösen felismerik egymást és szomszédossági kapcsolatot létesítenek. [1]
- **LSDB (kapcsolat-állapot adatbázis):** A LSDB az OSPF protokoll legfőbb adatszerkezete. Minden router saját adatbázisban tárolja a hálózatról szerzett topológiai információkat. Ezeket az információkat LSUpdate hálózati csomagok formájában szerzi meg.

  A router minden egyes, az LSUpdate csomag részeként kapott LSA (Link State Advertisement) csomagot feldolgoz és eltárol. Az új vagy frissített információkat tovább terjeszti a hálózatban. [1]

- **Dijkstra algoritmus:** Az program minden router esetében kiszámolja a hálózaton belüli legrövidebb utat az összes többi routerhez. Ehhez Dijkstra algoritmust használ, amely a kapcsolat-állapot adatbázis alapján meghatározza a legrövidebb utat egy súlyozott gráfban.

  Az eredményt egy egyszerűsített útvonalválasztási táblában tároljuk, valamint egy a terminálra is kiírja a topológia egy vizuális reprezentációját az adott router szempontjából

- **Naplózás:** A program működése során az összes fontos esemény és esetleges hibaüzenet routerenként naplózásra kerül. A Monitoring modulban található InfoLogger osztály .log fájlokat hoz létre, amik a futás során folyamatosan ír.

  A program futása során a LogMonitor figyeli ezeket a fájlokat és a fő terminálon megjeleníti a naplóbejegyzéseket.

- **Wireshark-kompatibilitás csomagrögzítés:** A program a küldött és fogadott csomagokat PCAP (Packet Capture) formátumú fájlokba menti. A program futása során folyamatosan interfészenként menti le a csomagokat.

A felhasználó a program leállítása után a Wireshark hálózati forgalom elemzővel részletesen elemezheti a hálózati kommunikációt.

2.3. A felhasznált módszerek

A projekt során a következő módszereket és eszközöket használtam:

- **Mininet** – Virtuális hálózatok létrehozására és szimulációjára.
- **Wireshark** – Hálózati forgalom elemzésére és naplózására.
- **Python** – A programozási nyelv, amellyel a teljes OSPF logika megvalósításra került.
- **Scapy** – Hálózati csomagok szimulációjára és manipulálására.
- **NetworkX** – A hálózati topológia ábrázolására és a legrövidebb utak kiszámítására.

2.4. A program használatához szükséges összes információ

2.4.1. Rendszerkövetelmények

- Python 3.10 vagy újabb
- Mininet (virtuális gépként vagy lokálisan telepítve)
- VirtualBox (ha a Mininet nem lokálisan van telepítve)
- Internetkapcsolat a csomagok letöltéséhez

2.4.2. Telepítési lehetőségek

A program futtatására kétféleképpen van lehetőség. Ha nem Ubuntu operációs rendszere van akkor VirtualBox letöltésével és egy Mininet virtuális gép használatával van lehetőség a program működtetésére.

A) Mininet VM + VirtualBox

1. Töltse le a Mininet VM-et a Mininet hivatalos oldaláról.
2. Importálja VirtualBox-ba a telepítéssegítő instrukciók alapján.
3. További beállítások:
   - **Settings > System > Processor > Enable PAE/NX**
   - **Settings > Network -> Adapter 1 > Attached To > Bridged Adapter**
4. Indítsa el, majd jelentkezzen be az oldalon megadott adatokkal.
5. Telepítse és állítsa be a virtuális Python környezetet:
   5.1. Futtassa le a következő parancsokat a Python 3.10 telepítéséhez:

```
$ sudo apt update

$ sudo apt install software-properties-common -y

$ sudo add-apt-repository ppa:deadsnakes/ppa -y

$ sudo apt update
```

```
$ sudo apt install python3.10 python3.10-venv python3.10-dev -y
```

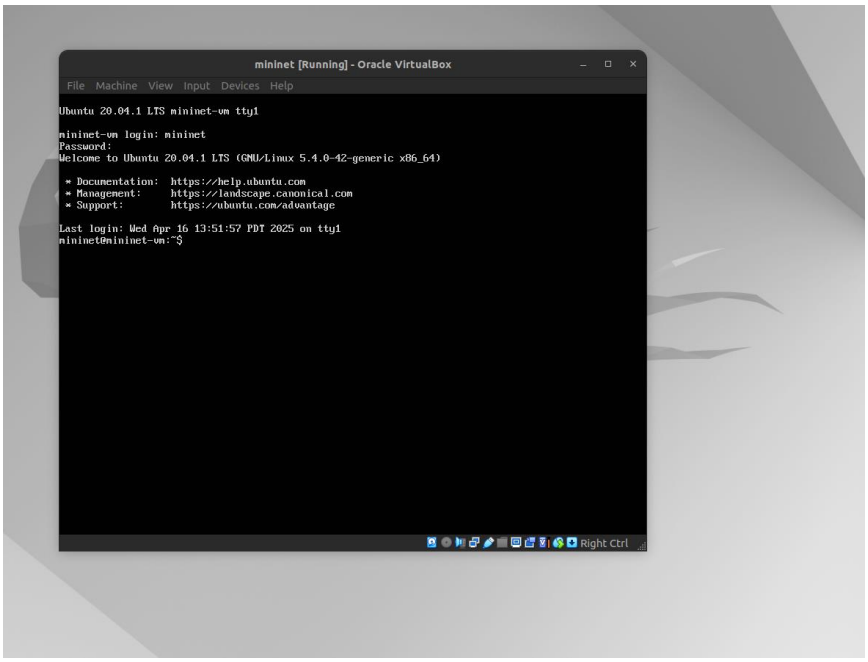
5.2. Indítson el egy Python 3.10 virtuális környezetet:

```
python3.10 -m venv [környezet neve]

source [környezet neve]/bin/activate
```

5.3. Telepítse a szükséges Python csomagokat:

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



2. ábra: Mininet VM Virtualbox-ban futtatva

B) Lokális Linux rendszerre letöltés

A program Python 3.10-ben íródott, ha a Linux alapból régebbi verziót futtat kérem a Python verziót itt is frissítse az előző pont alapján.

1. Töltse le a Mininet-et lokálisan a hivatalos oldal alapján.
2. Indítson egy Python 3.10 virtuális környezetet.
3. Töltse le a szükséges csomagokat:

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

2.4.3. Csatlakozás és fájlmásolás

A Mininet VM-használatának egy felhasználóbarátabb módja, ha a felhasználó a SSH csatlakozással saját terminálból használja a virtuális gépet. A következő parancsok mindegyikét saját terminálból futtassa.

SSH-val csatlakozzon a Mininet VM-hoz:

```
$ ssh mininet@[IP-cím]
```

Másolja be a projektfájlokat:

```
$ scp -r Downloads/thesis mininet@[IP-cím]:~
```

2.4.4. A program futtatása

A programot felhasználási módszertől függően kétféleképpen lehet futtatni. **Automatikus** indítás során az OSPF folyamat a háttérben fut és az eseményekről készült naplóüzenetek a fő terminálba íródnak ki. **Manuális** indítás során a felhasználó külön terminált nyit azoknak a routereknek, ahol a folyamatot futtatni szeretné és manuálisan indítja el az OSPF programot.

A kód több helyen is rendszerfájlt olvas, ezért a **sudo** használata mindkét futtatási módszernél fontos.

A) Automatikus (alapértelmezett)

```
$ sudo python3 run.py auto
```

B) Manuális (külön terminál minden routerhez)

```
$ sudo python3 run.py manual

$ xterm [router neve]

$ sudo python3 start_ospf.py [router neve]
```

2.4.5. Wireshark használata

A **`packet_logs/`** könyvtárban található PCAP naplófájlok tartalmazzák a routerek interfészein küldött hálózati csomagokat. Ezeknek elemzéséhez a Wireshark hálózati forgalom elemző program használatát ajánlom:

1. Telepítse a Wiresharkot a hivatalos oldalról.
2. Másolja át a naplózott **`.pcap`** fájlokat.

```
$ scp -r mininet@[IP-cím]:~/thesis/packet_logs ~/Downloads
```

3. Nyissa meg a kívánt fájlt Wireshark-ban: **`[interfész neve].pcap`**

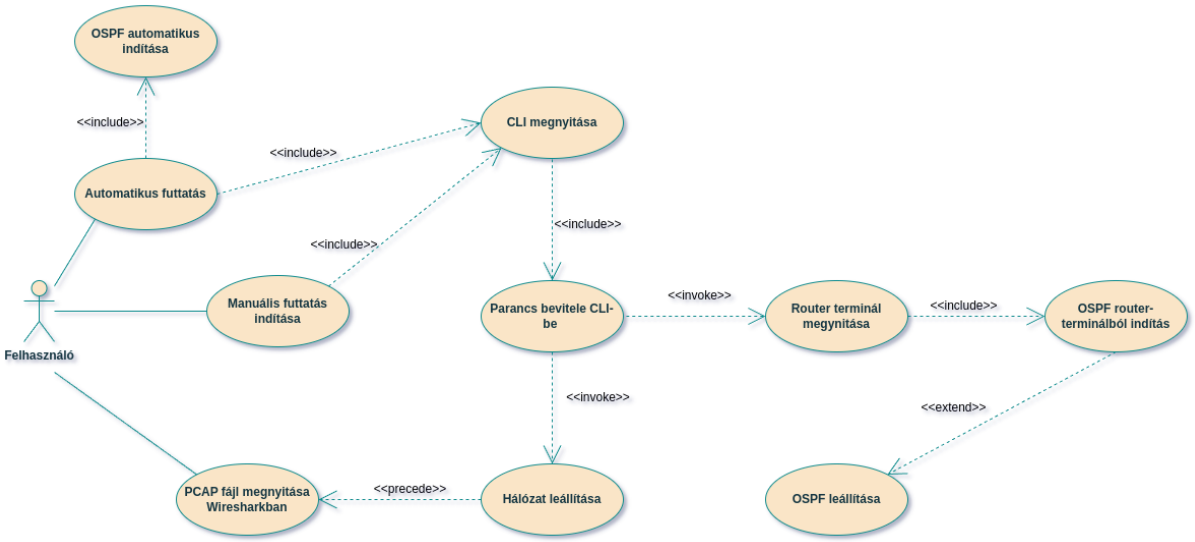
3. Fejlesztői dokumentáció

3.1. A probléma részletes specifikációja

A projekt célja egy egyszerűsített, oktatási célú OSPF (Open Shortest Path First) protokoll szimulációjának megvalósítása Python nyelven, Mininet alapú virtuális hálózat környezetben. A program feladata, hogy:

- Szimulálja az OSPF Hello protokollt és a szomszédok közötti kapcsolatfelvételt.
- Leírja a szomszédossági állapotok változását.
- Rögzítse a hálózati eseményeket logfájlokba.
- Rögzítse a hálózati csomagokat PCAP formátumú fájlokba.
- Vizualizálhatóvá tegye a topológia felépülését.

3.1.1. Használati esetek



3. ábra: Használati eset diagram

3.2. Felhasznált módszerek

A program Python nyelven készült, moduláris, objektumorientált megközelítéssel. Az alkalmazott eszközök és technológiák:

- **Python 3.10** – a programozási nyelv.
- **Mininet 2.3.0** – virtuális hálózati környezet létrehozására.
- **Scapy** – hálózati csomagok kezelésére és elemzésére.
- **Wireshark** – a PCAP fájlok megtekintésére és elemzésére.
- **Pytest** – automatikus tesztelésekhez.
- **YAML** – konfigurációs fájlformátum a hálózati elemek leírására.
- **PCAP formátum** – hálózati forgalom rögzítésére.

4. ábra: A program nyelve

Az adatok kezelésére és tárolására a következő struktúrák kerültek kialakításra:

- YAML konfigurációs fájlok a hálózati topológia leírásához és a hálózati elemek konfigurációjához.
- PCAP fájlok a routerek interfészein küldött és fogadott hálózati csomagok rögzítéséhez.
- Szöveges logfájlok az események és állapotváltozások dokumentálására.

3.2.1. Verziókövetés és forráskód

A szakdolgozat fejlesztése teljes egészében verziókövetéssel történt, a Git és a GitHub platform segítségével. A fejlesztés lépései nyomon követhetők a **thesis** GitHub repository fő ágán (main branch), amely tartalmazza a legfrissebb, stabil állapotú forráskódot.

A forráskód elérhető és letölthető a következő linken: **https://github.com/x3gan/thesis** (main branch)

3.3. Hasznos fogalmak

3.3.1. Alap fogalmak:

- **OSPF (Open Shortest Path First):** Szabványosított kapcsolat-állapot alapú útvonalválasztó protokoll, amely az IP hálózatokban használnak a legoptimálisabb útvonalakat kiszámítására. [1]
- **Kapcsolat-állapot protokoll:** Olyan útvonalválasztó protokoll, amely az egész hálózat topológiájának ismeretében számít útvonalakat. A routerek minden ismert kapcsolatról információt cserélnek egymással, ezáltal minden résztvevő ugyanazt a topológiát látja.
- **Router:** Olyan hálózati eszköz, amely különböző hálózatok között továbbítja adatcsomagokat.
- **Hálózati topológia:** A hálózatot alkotó eszközök (pl. routerek, switchek, hosztok) fizikai vagy logikai elrendezése.
- **Dijkstra algoritmus:** Olyan gráfelméleti algoritmus, ami egy adott csúcsból kiindulva meghatározza a legrövidebb utat minden más csúcshoz.

3.3.2. Protokoll specifikus fogalmak:

- **Hello üzenet:** Az OSPF által használt üzenetforma, amelyet a routerek rendszeres időközönként küldenek egy adott interfészen. Célja a szomszédos routerek felfedezése

és szomszédsági kapcsolatok fenntartása. Az üzenetek tartalmazzák a szomszédok listáját és az időzítő paramétereket.

- **Hello Interval:** Az az időintervallum, amelyen belül egy router rendszeresen Hello csomagot küld minden aktív interfészen. Alapértelmezett értéke: 10 másodperc. Ahhoz, hogy két router kapcsolatot építsen fel egymás között, a Hello Interval értékeknek meg kell egyeznie.
- **Dead Interval:** Az az időintervallum, amelyen belül, ha egy router nem kap Hello csomagot a szomszédjától, akkor azt a szomszédot elérhetetlennek véli és Down állapotba helyezi a kapcsolatot.
- **RID (Router ID):** Az OSPF-ben résztvevő routerek egyedi azonosítója.
- **Area ID:** Az OSPF hálózatot nagyméretű topológia esetén logikai területekre lehet osztani. A területek célja a skálázhatóság és adatforgalom csökkentése. Az Area ID jelzi, melyik router melyik területhez tartozik.
- **LSA (Link State Advertisement):** Az OSPF által használt üzenet, amely a routerek link-információját terjeszti a hálózatban.
- **LSDB (Link State Database):** Az OSPF routerek által karbantartott adatbázis, amely a hálózat teljes topológiai információját tartalmazza.
- **Neighbor (Szomszéd):** Olyan router, amely közvetlenül elérhető egy adott interfészen keresztül, és amelyről Hello üzeneteket kapunk.

(RFC 2328 [1])

3.3.3. Technikai fogalmak:

- **Mininet:** Egy hálózatszimulációs eszköz, amely lehetővé teszi virtuális hálózati topológiák gyors létrehozását Linux környezetben és támogatja a Python-nal való fejlesztést is.
- **Scapy:** Egy Python könyvtár, amely lehetővé teszi hálózati csomagok kézi létrehozását, küldését és fogadását. Emellett az támogatja az OSPF csomagtípusokat.
- **Pcap fájl:** Egy hálózati forgalom rögzítésére szolgáló fájlformátum. Ezeket a típusú fájlokat különböző hálózati elemzőkkel lehet megnyitni.
- **NetworkX:** Egy Python könyvtár, amely grafikonok (hálózatok) létrehozására, rajzolására és elemzésére szolgál. Ennek a segítségével közérthetően megjelenítve lehet a topológiákat ábrázolni.

3.4. A szoftver logikai és fizikai szerkezete

3.4.1. Logikai architektúra

A program logikai szempontból három fő modulra bontható fel:

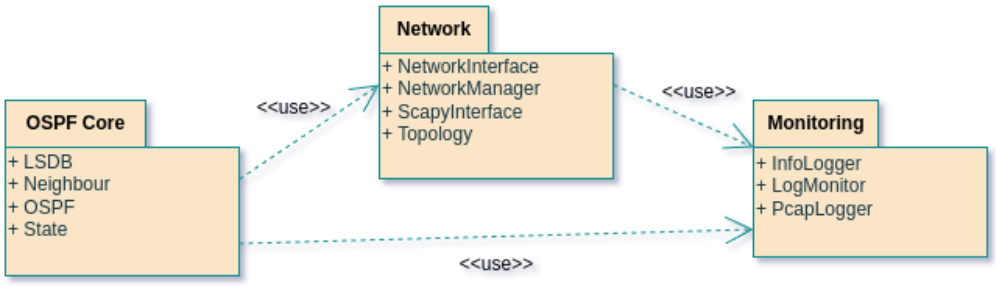
1. **Hálózatkezelő modul (Network)**
    a. Felépíti a virtuális hálózatot Mininet segítségével.
    b. Elindítja routereken futó OSPF folyamatokat.
    c. Összefogja a különböző modulok működését.

2. **OSPF Core modul**
    a. Szimulálja az OSPF Hello protokollt.
    b. Kezeli a szomszédállapotok változásait.
    c. LSDB-t (Link-State Database) épít és karbantart.
    d. Útvonalválasztási táblát készít és állít be.
3. **Monitoring modul**
    a. A logfájlok és PCAP állományok kezeléséért felelős.
    b. Folyamatosan követi a hálózati kommunikációt.
    c. Figyeli a naplófájlokat és megjeleníti a routerek tevékenységeit.

További elemek:

- **`common/`** könyvtár: Itt tárolom a több osztály és modul által használt segédfüggvényeket, például a YAML konfigurációs fájlok parse-olására használt **`get_config()`** függvényt.
- **`tests/`** könyvtár: egységtesztek tárolása.

3.4.2. Fizikai architektúra



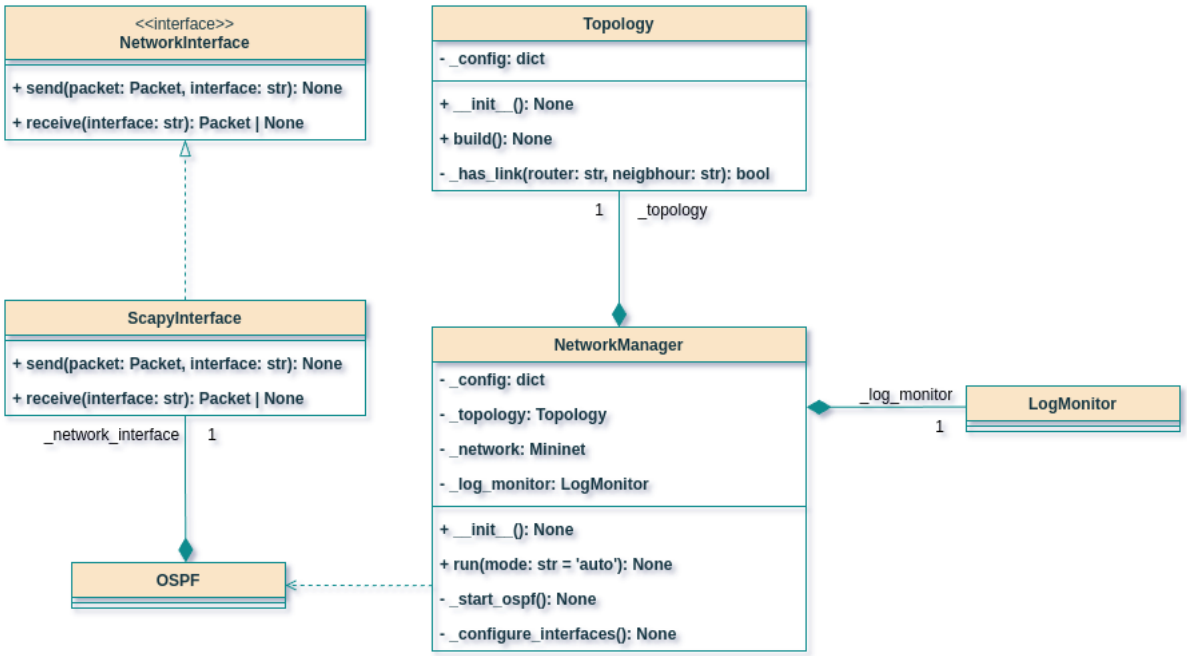
5. ábra: Csomagdiagram

3.5. A megvalósított modell

Mind a három modul különböző osztályokból épül fel, ezeknek az osztályszintű leírása a következőkben olvasható. A metódusok részletes leírása a kódfájlokban, metódusonként **docstring** formájában olvashatók.

3.5.1. Network modul osztályai

A Network modul tartalmazza azokat az osztályokat, amelyek a hálózat fizikai szintjével közvetlenül kommunikálnak. Feladatuk az interfészek kezelése, a hálózati csomagok küldése és fogadása, illetve a hálózati topológia dinamikus felépítése és működtetése a szimuláció során.



6. ábra: Network osztálydiagram

3.5.1.1. Class: NetworkManager

A **`NetworkManager`** osztály a Network modul központi eleme, amely koordinálja a hálózati komponensek működését.

Fő feladatai:

- Betölti a hálózati topológia konfigurációját a YAML konfigurációs fájlból.
- Az egyedi topológiát a **`Topology`** osztály segítségével építi fel, majd ezt átadja a Mininet virtuális hálózatnak.
- Létrehozza a Mininet hálózatot, elindítja, majd a szimuláció futása végén leállítja.
- A hálózat elindítása után a **`configure_interfaces()`** metódussal beállítja a routerek interfészeit a konfigurációs fájlban megadott IP-címekkel.
- Példányosítja a Monitoring modul **`LogMonitor`** osztályát, amely a szinte valós idejű esemény megfigyelésért felel és elindítja azt.
- Miután minden más elindul elindítja az OSPF folyamatokat a routereken és ha “manual” módban indul, elindít egy parancssort a hálózati eszközökkel való kommunikáláshoz.

A **`NetworkManager`** összefogja a szimuláció összes elemét, biztosítva az összehangolt működést.

3.5.1.2. Class: NetworkInterface

A **`NetworkInterface`** osztály egy absztrakt osztály a hálózati interfészeken történő kommunikáció kezelésére.

Fő funkciói:

- Hálózati csomagok küldésének és fogadásának kezelése az interfészeken.

- Interfészműveletek biztosítása, amelyet az **`OSPF`** osztály használ.

A **`NetworkInterface`** nem végez tényleges csomagkezelést, hanem egy általános interfészt ad ezeknek a működéséhez.

3.5.1.3. Class: ScapyInterface

A **`ScapyInterface`** osztály a **`NetworkInterface`** megvalósítása, amely Python Scapy könyvtárát használja a hálózati csomagküldésre és-fogadásra.

Fő funkciói:

- A **`sendp()`** függvényt alkalmazva képes hálózati csomagokat az adott interfészeken keresztül küldeni.
- A **`sniff()`** függvényt használja az adott interfészen érkező csomagok fogadására és feldolgozására.

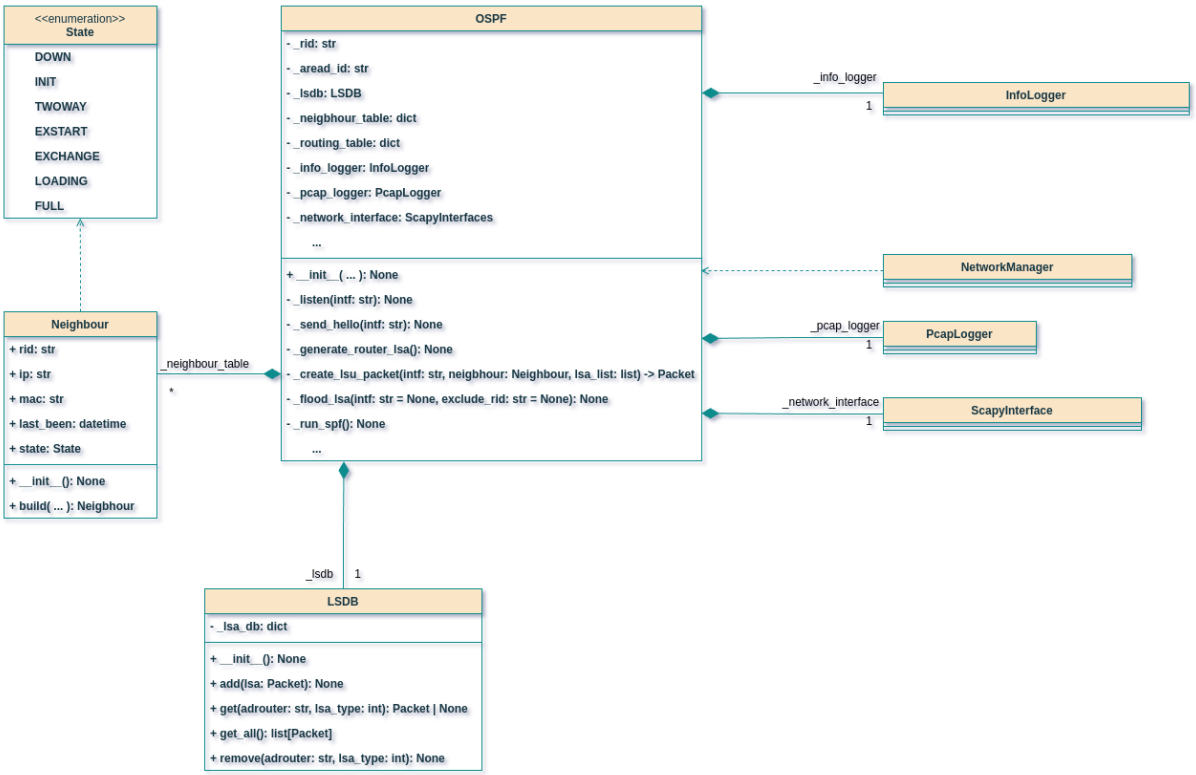
A **`ScapyInterface`** biztosítja az OSPF protokoll számára a hálózati kommunikációs funkciókat anélkül, hogy annak közvetlen kelljen a hálózati szinttel foglalkoznia.

3.5.2. OSPF Core modul osztályai

Az OSPF Core modul tartalmazza az OSPF algoritmusának szimulált működéséhez szükséges összes főbb adatstruktúrát, logikát és folyamatot. A modul célja, hogy a routerek közti útvonalválasztást valós időben, párhuzamos szálkezeléssel szimulálja.

A modul központi eleme az **`OSPF`** osztály, ami az algoritmus működésének minden funkcióját magába foglalja. Emellett a modul tartalmaz, egy kapcsolat-állapot adatbázist (**`LSDB`**), szomszédot reprezentáló objektumot (**`Neighbour`**), valamit állapotokat definiáló felsoroló típust (**`State`**) is.

(RFC 2328 [1])



7. ábra: OSPF Core osztálydiagram

3.5.2.1. Class: OSPF

Az OSPF egy összetett osztály, ami egyben kezeli az OSPF által generált hálózati adatok feldolgozását, a hálózati csomagok küldését az adatbázis karbantartását, valamint a legrövidebb út kiszámításáért majd a topológia megjelenítését.

A kód átláthatósága és a könnyebb karbantarthatóság érdekében az **`OSPF`** osztály logikai szekcióra van bontva.

Fő szekciók:

Hálózati csomagkezelés:

- A **`ListenerThread`** szál, a **`_listen()`** metódust használja fel, ami a **`ScapyInterface`** segítségével folyamatosan figyeli a beérkező csomagokat és beleteszi azokat egy csomagfeldolgozó sorba.
- A **`PacketProcessorThread`** szál a **`_process_packet()`** metódus segítségével figyeli csomagfeldolgozó sort és az ide belekerülő csomagokat típustól függően további feldolgozásra küldi.

Hello csomagok kezelése:

- A **`HelloThread`** szál a**`_send_hello()`** metódust felhasználva, 10 másodpercenként multicast Hello csomagokat küld, minden nem Down állapotban lévő szomszéd adatait eltárolva.

- A **`_process_hello`** metódus a sorból kivett csomagokat feldolgozza és a kapott csomag alapján frissíti a szomszédállapotot a **`_neighbour_table`** szótárban.

LSUpdate csomagok kezelése:

- A **`_generate_router_lsa()`** metódus a Router LSA-t készít annak a szomszédnak az adataival. Ezt az LSA-t utána eltárolja a **`_lsdb`** kapcsolat-állapot adatbázisában és elindítja az LSA-k terjesztését.
- A **`_flood_lsa()`** a FULL állapotban lévő szomszédoknak küldi az LSA csomagokat.
- A fogadott LSUpdate csomagokat a **`_process_lsa()`** feldolgozza és szekvenciaszám alapján a frissebb LSA-kat eltárolja az LSDB-ben, majd tovább terjeszti.

Szomszédok állapotának kezelése:

- A **`StateWatchThread`** szál a **`_state_watch()`** metódussal figyeli a szomszédokat és kezeli az állapotváltozásokat.
- A **`Neighbour`** objektum állapotának változása alapján különböző műveleteket hajt végre. (pl. **`_generate_ruter_lsa()`**)
- A **`DownThread`** szál az **`_is_down()`** állapotfigyelő metódust használja, hogy ellenőrizze a szomszédtól érkezett-e Hello a Dead Interval 40 másodperces időtartamán belül, és ha szükséges leválasztja a szomszédot és tovább terjeszti ezt az információt.

Topológia és legrövidebb út számítása:

- A **`_run_spf()`** metódus az LSDB alapján meghatározza a teljes hálózati gráfot.
- A NetworkX könyvtárral Dijkstra-alapú legrövidebb út számítás történik. Az eredmény alapján frissül a **`_routing_table`**.
- Vizuálisan megjeleníti a hálózat topológiát a terminálban és terminál parancsokat használva beállítja a routerek útvonalválasztási tábláját.

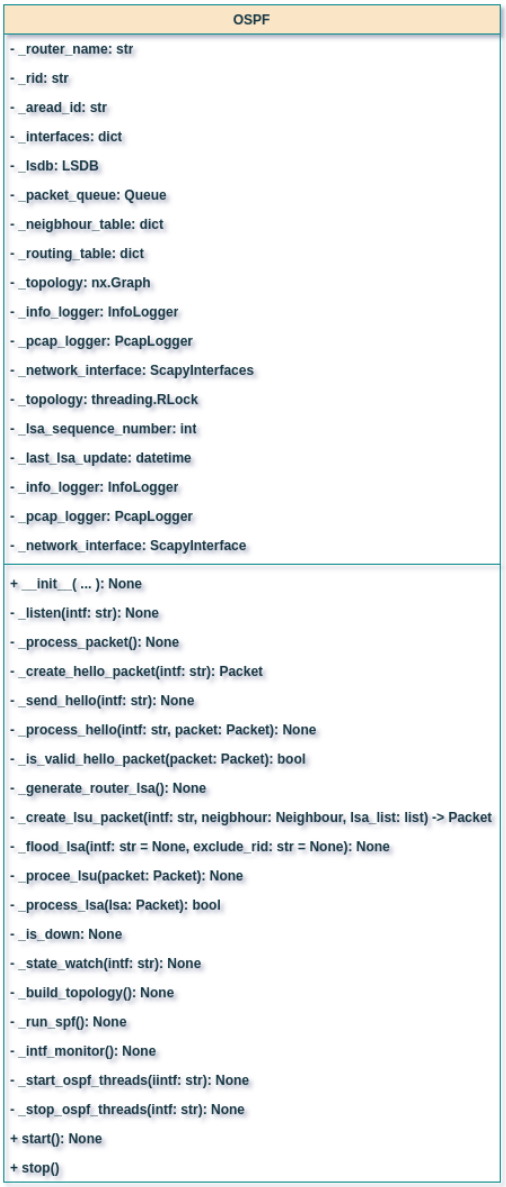
Életciklus-kezelés:

- A szálak futtatása a **`threading.Thread`** osztályon alapul, futtatását a **`start()`** metódus indítja.
- A **`stop()`** metódus beállítja a **`_stop_event`** eseményt. Ezt minden szál periodikusan ellenőriz, ezzel biztosítva, hogy 1-2 másodpercen belül leálljanak.

Az **`InfoLogger`** segítségével minden fontos állapotváltozás, esemény, szomszédkapcsolat és LSA terjesztés naplózásra kerül routerenként külön **`.log`** fájlba. Ezeket a fájlokat automatikus futtatás során a **`NetworkManager`** **`LogMonitor`**-ja folyamatosan ellenőrzi és a felhasználó számára megjeleníti.

A **`PcapLogger`** segítségével minden küldött és fogadott OSPF hálózati csomag **`PCAP`** formátumban is mentésre kerül.

Fontos! A naplófájlok új OSPF indításakor routerenként automatikusan törlődnek, kivéve, ha manuálisan archiválásra kerülnek.



8. ábra: OSPF osztálydiagram

([RFC 2328 [1]])

3.5.2.2. Class: LSDB

A LSDB a router kapcsolat-állapot adatbázisának egy reprezentációja, ami az LSA-kat típusonként tárolja egy listában.

Főbb funkciói:

- Új LSA hozzáadása az adatbázishoz, LSA típusonként tárolva.
- Adatbázisból LSA lekérése RID alapján.

- Össze eltárolt LSA lekérése az adatbázisból.
- Kapcsolat elvesztése esetén pedig ki is lehet törölni LSA-t az adatbázisból.

Ezek a műveletek elengedhetetlenek az adatbázis karbantarthatósága érdekében.

(RFC 2328 [1])

3.5.2.3. Class: Neighbour

Minden OSPF folyamat **`_neighbour_table`** szótára interfészenként egy listában tárolja az **`Neighbour`** objektumokat. Ezeket az objektumokat a felfedezett szomszéd adati alapján hozza létre. Egy-egy **`Neighbour`** példány a következő adatokat tárolja a szomszédról:

- RID, IP-cím, MAC-cím,
- Utolsó Hello kommunikáció időpontja,
- A szomszéd aktuális állapota (**`State`**).

A **`Neighbour`** osztály fontos szerepet játszik a szükséges állapotkezelésben és az útvonalválasztási tábla kiépítésében.

(RFC 2328 [1])

3.5.2.4. Class: State

A **`State`** egy felsoroló (**`Enum`**) osztály, amely az OSPF által használt állapotokat tartalmazza:

- Down, Init, TwoWay, ExStart, Exchange, Loading, Full.

Mivel a **`Neighbour`** objektumok eltárolják az állapotot, ezért az OSPF folyamatok alatti állapotváltás logikusan végigkövethető.

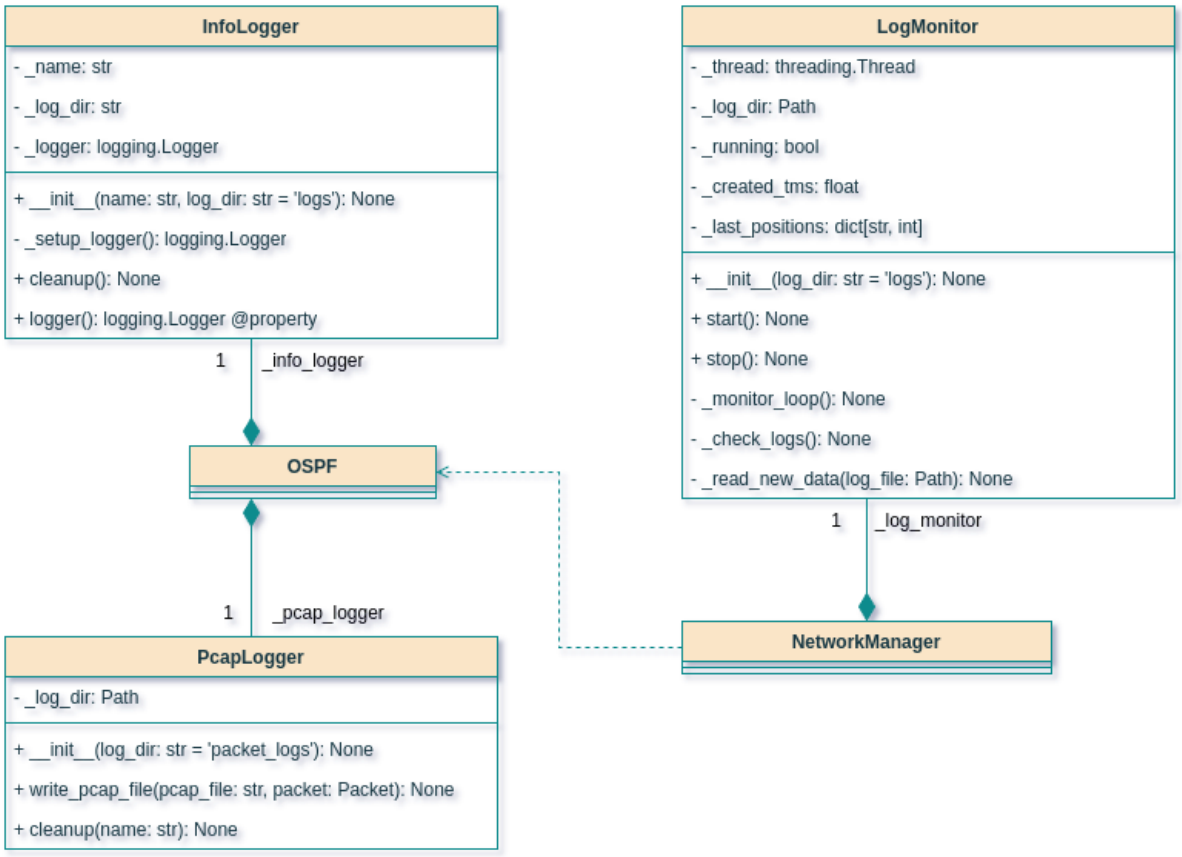
(RFC 2328 [1])

3.5.2.5. OSPF leállítása

Az OSPF helyes működése érdekében a folyamatok egy leállítási parancsig futnak, ezért a folyamatok futását mindig **`CTRL+C`** segítségével érdemes leállítani. Ezt minden szál érzékeli a **`_stop_event`** eseményen keresztül és megkezdik a szabályos leállást.

3.5.3. Monitoring modul osztályai

A Monitoring modul a hálózati események naplózásáért és azok valós idejű megjelenítéséért felelős. Itt találhatóak az OSPF működéséhez kapcsolódó naplózási és hálózati forgalom-mentési osztályok, valamint a naplófájlokat megfigyelő háttérfolyamat is.



9. ábra: Monitoring osztálydiagram

3.5.3.1. Class: InfoLogger

Az **`InfoLogger`** osztály a Python **`logging.Logger`** osztályára épül, és kibővíti azt egyedi, időbélyegzővel ellátott üzenetformátummal.

Főbb funkciói:

- Beállítja az üzenetek rögzítését fájlba és konzolra egyaránt.
- Támogatja az **`INFO`**, **`WARNING`** és **`ERROR`** bejegyzési szinteket.
- A naplófájlok routerenként kerülnek eltárolásra, a **`logs/`** könyvtárba, a router nevével ellátva menti.
- Az **`InfoLogger`** példányosításakor automatikusan kitörli az adott routerhez tartozó korábbi futásból származó naplófájlokat a **`cleanup()`** metódus segítségével.

Ezáltal minden futás új naplófájlokkal indul, elkerülve a régi adatok keveredését az aktuális futás naplóival.

3.5.3.2. Class: PcapLogger

A **`PcapLogger`** osztály a Scapy **`PcapWriter`** objektumát használja a hálózati forgalom rögzítésére PCAP formátumban.

Főbb funkciói:

- A routerek interfészein küldött és fogadott hálózati csomagokat **`.pcap`** formátumú fájlokba menti.

- Az csomagokat a **packet_logs/** könyvtárba helyezi el, interfészenként külön fájlokba.
- A naplózási fájlok tisztítását a példányosítás során a **cleanup()** metódus végzi.

Ezzel a felhasználó a program futása után Wireshark segítségével részletesen elemezheti az elküldött és fogadott hálózati csomagokat.

3.5.3.3. Class: LogMonitor

A **LogMonitor** egy háttérszálon futó folyamat, amely folyamatosan figyeli a **logs/** könyvtárban lévő naplófájlokat.

Főbb funkciói:

- A program indulásakor eltárolja a létrehozásának időpontját, és csak az ezután létrehozott fájlokat figyeli meg.
- 100 milliszekundumos időközönként ellenőrzi a logfájlok méretét.
- Ha egy fájl mérete nő, akkor csak a korábban eltárolt pozíció és az aktuális fájlméret közötti új adatot olvassa be és írja ki a konzolra.
- A megfigyelt fájlok utolsó ismert olvasási pozícióját a **_last_positions** szótárban tárolja, fájlnév kulcs szerint. A fájloknak a kezdeti ismert pozíciója 0.

Ez a megoldás lehetővé teszi, hogy a felhasználó a főterminálon szinte valós időben követhesse a routerek közötti kommunikáció főbb eseményeit és állapotváltozásait.

Mivel a **LogMonitor** már az OSPF folyamatok elindulása előtt létrejön (ennek során törlődnének a korábbi naplófájlok), különösen fontos a helyes indítási sorrend: csak az aktuális szimuláció során keletkező naplófájlokat szabad, hogy figyelembe vegye.

3.6. Naplófájlok és felépítésük

A program működése során két különböző típusú naplózási rendszer kerül alkalmazásra:

- **logs/** könyvtár: A routerek állapotváltozásairól szóló naplókat tárolja, például a szomszédkapcsolatok kiépülését vagy szétkapcsolódását. Emellett rögzíti a csomagküldések és fogadások eseményeit is. Az OSPF protokoll futása során, ha egy szomszéd elérhetetlenné válik, erről figyelmeztető üzenet kerül rögzítésre. Ha a futás során bármilyen hibába ütközik a program, annak üzenete szintén ebben a könyvtárban jelenik meg.

  Minden router saját nevén, külön fájlban tárolja ezeket az eseményeket, amelyeket a **LogMonitor** szinte valós időben jelenít meg a terminálon.

- **packet_logs/** könyvtár: Az interfészek közötti hálózati kommunikációt **PCAP** formátumban tárolja. A fájlok a következő névformátummal jönnek létre: **[RouterNév]-[InterfészNév].pcap**. Ezek a fájlok a szimuláció végén másolhatók le a Mininet gépről, és Wireshark-kal részletesen elemezhetők.

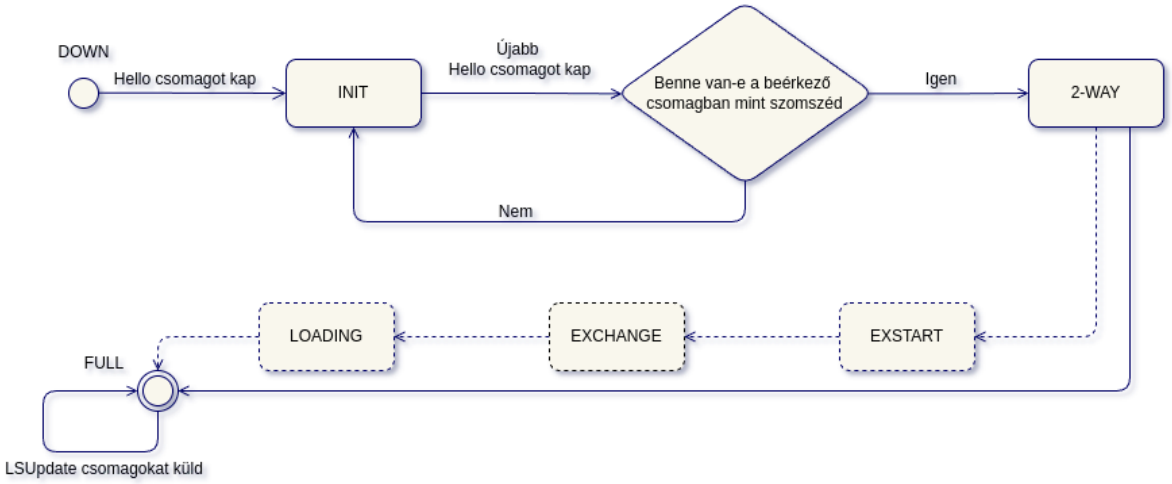
A **PCAP** fájlok lehetővé teszik a szimulált OSPF protokoll mélyebb szintű, csomagszintű vizsgálatát és a hálózati viselkedés részletes elemzését.

3.7. OSPF szomszéd állapotok (Neighbor States)

Az OSPF protokoll működése során a routerek minden interfészükön külön-külön tartják nyilván, hogy milyen állapotban van az adott szomszéd. Az állapotok egymás után következnek a szomszédossági kapcsolat kiépülése során. Az állapotok célja, hogy kiépítse a szomszédosságot és szinkronizálja a kapcsolat-állapot adatbázist.

Az állapotváltásokat a **State** felsoroló reprezentálja a programban és az **OSPF** osztály szomszédkezelő logikáján keresztül frissülnek.

A routerek szomszédos interfészei az alábbi állapotokon mennek át:



10. ábra: OSPF állapotdiagram

- **Down:** Kezdeti állapot. Minden új vagy hosszú ideje inaktív interfészre vonatkozik. Azt jelzi, hogy a router még nem kapott Hello csomagot vagy már volt korábbi kapcsolat, de Dead Interval (40 másodperc) elteltével nem érkezett Hello csomag a szomszédtól.
- **Init:** A router Hello csomagot küldött a multicast címre, viszont még nem kapott választ. A router a Hello csomagban elküldi a saját adatait és azt is milyen szomszédokat ismer.
- **2-Way:** Kölcsönös felismerést jelző állapot. A router olyan Hello csomagot kapott a szomszédjától, aminek a szomszéd listájában felismeri a saját RID-ját. Ebben azz állapotban történik meg annak az eldöntik, hogy teljes szinkronizációs kapcsolatot építenek ki.
- **ExStart, Exchange, Loading:** Ezek az állapotok a kapcsolat-állapot adatbázis szinkronizációs lépések. A routerek egymás között eldöntik, ki a “master” és ki a “slave”, majd fokozatosan kicserélik a tárolt LSA-kat.

A jelenlegi implementációban ezek az állapotok egyelőre csak reprezentációs céllal szerepelnek. Az adatszinkronizációs lépéseket a program egyszerűsített módon, **`2-Way -> Full`** állapotváltással hajtja végre.

- **Full:** Az OSPF szomszédossági kapcsolat utolsó állapota. A router és a szomszéd között a kapcsolat-állapot adatbázis teljesen szinkronizált. A saját LSA-ját elküldte a szomszédnak. Minden frissített vagy új LSA-t ebben az állapotban tovább terjeszti.

(RFC 2328 [1])

3.8. 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`** nélkül is lefutnak, azonban így a Mininet-et használó teszteseteket a program átlépi, hiszen a Mininet futása **`sudo`** jogosultságot igényel.

3.8.1. Tesztkörnyezet

- **Operációs rendszer:** Ubuntu 22.04 LTS
- **Python verzió:** 3.10
- **Mininet verzió:** 2.3.0
- **Python könyvtárak:** requirements.txt-ben rögzítve

3.8.2. OSPF Core modul tesztelése

3.8.2.1. 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.

3.8.2.2. Teszteset: Létrejön egy OSPF Hello csomag

Bemenet	OSPF osztály interfész adatokat kap, hogy csomagot tudjon létrehozni.

Elvárt kimenet	Az interfészadatok alapján létrejön az OSPF Hello csomag az elvárt adatokkal.
Tényleges kimenet	A megadott adatok alapján és az osztály konstans adatai alapján létrejön a Hello csomag.

3.8.2.3. Teszteset: A beérkezett OSPF Hello csomagot feldolgozza

Bemenet	Az osztály Hello csomag feldolgozója kap egy csomagot.
Elvárt kimenet	A csomagban lévő adatokat jól értelmezi és eltárolja a szomszédokat tároló szótárban.
Tényleges kimenet	Jól értelmezi a csomagban található adatokat és elhelyezi a szomszédokat tároló szótárban a megfelelő interfészen.

3.8.2.4. Teszteset: LSA létrehozása

Bemenet	Az OSPF osztály kap egy Full állapotban lévő szomszédot a szomszédot tároló szótárba.
Elvárt kimenet	Generál Router LSA-t, ami a Full állapotban lévő szomszédokat tartalmazza, majd beleteszi az LSDB-be.
Tényleges kimenet	A Router LSA belekerül az LSDB-be és kiolvasás után a megadott jó adatokat tárolja.

3.8.2.5. Teszteset: LSU csomag létrehozása

Bemenet	Az osztály kap szomszéd és Router LSA adatokat, amit beleteszünk az LSDB-be.
Elvárt kimenet	Létrehozza az LSU csomagot a adatokkal és az LSA-val az LSA listájában, amit az LSDB-ből olvas ki.
Tényleges kimenet	A megfelelő adatokkal létrehozza az LSU csomagot. A csomag LSU listájában az az LSA van, amit az LSDB-ből olvasott ki.

3.8.2.6. Teszteset: LSU-ban lévő LSA csomag feldolgozása

Bemenet	Az LSA csomagfeldolgozó kap egy előre előkészített LSA csomagot.
Elvárt kimenet	A feldolgozó folyamat feldolgozza a csomagot és beleteszi a Router LSDB-jébe. Jól dolgozza fel az adatokat és jól tárolja el.
Tényleges kimenet	Megfelelően feldolgozza az adatokat az előre készített LSA alapján. Az LSDB-ben a megfelelő adatokkal tárolja el.

3.8.3. Network modul tesztelése

3.8.3.1. 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.

3.8.3.2. 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.

3.8.3.3. 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.

3.8.3.4. 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.

3.8.3.5. 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.

3.8.4. Tesztelési bizonyíték

```
lilexuhr@eh:~/PycharmProjects/thesis$ sudo pytest -v tests
========================================= test session starts ==========================================
platform linux -- Python 3.10.12, pytest-8.3.5, pluggy-1.5.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/lilexuhr/PycharmProjects/thesis
collected 11 items

tests/network_test.py::test_empty_topology PASSED                                              [  9%]
tests/network_test.py::test_config_topology PASSED                                             [ 18%]
tests/network_test.py::test_network_manager_create PASSED                                      [ 27%]
tests/network_test.py::test_router_network_manager PASSED                                      [ 36%]
tests/network_test.py::test_ospf_start PASSED                                                  [ 45%]
tests/ospf_test.py::test_ospf_initialization PASSED                                            [ 54%]
tests/ospf_test.py::test_hello_packet_creation PASSED                                          [ 63%]
tests/ospf_test.py::test_hello_packet_processing PASSED                                        [ 72%]
tests/ospf_test.py::test_lsa_packet_creation PASSED                                            [ 81%]
tests/ospf_test.py::test_lsu_packet_creation PASSED                                            [ 90%]
tests/ospf_test.py::test_lsa_packet_processing PASSED                                          [100%]
```

11 .ábra: Tesztelési bizonyíték (pytest)

4. További fejlesztési lehetőségek

A program egyszerűsítve 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 kommunikáció útvonalát.

Teljes OSPF-állapot implementáció: A program jelenleg az algoritmus működéséhez elengedhetetlen állapotokat implementálja, azonban ez továbbfejleszthető lenne, amivel valósághűbb kapcsolat-állapot adatbázis szinkronizációt lehetne szimulálni.

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.

5. Irodalomjegyzék

[1] J. Moy, OSPF Version 2, RFC 2328, 1998. https://datatracker.ietf.org/doc/html/rfc2328, 2025.05.01.