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.

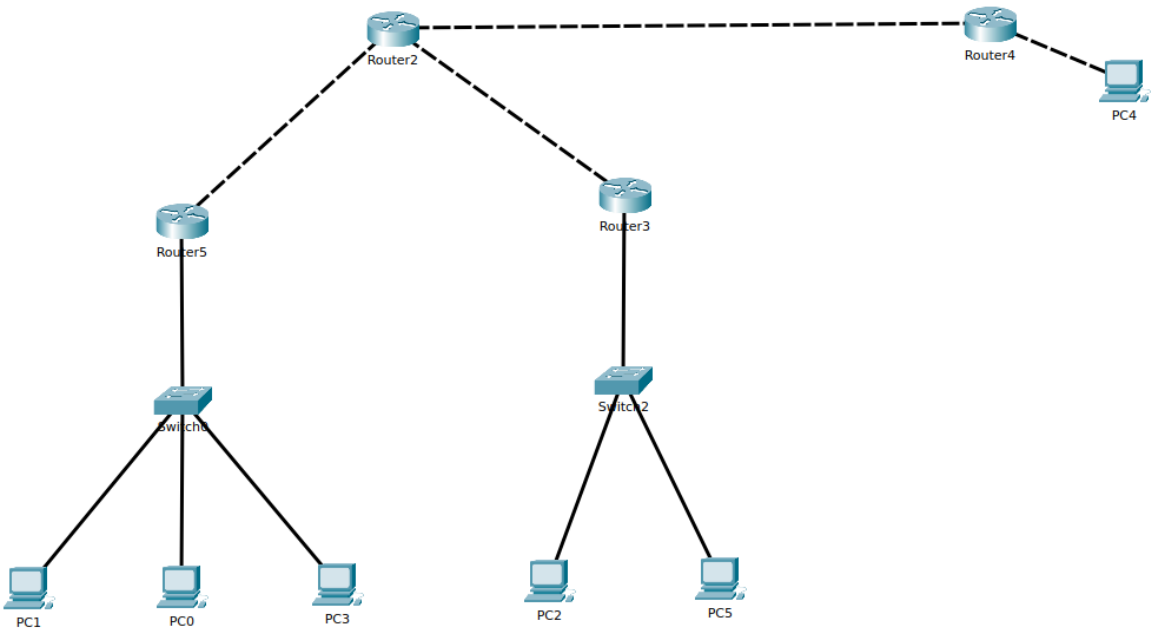
Budapest, 2024. 10. 15.

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 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):** Felfedezi a szomszédos routereket.
- **LSDB (kapcsolat-állapot adatbázis):** Frissül minden új LSA csomag alapján.
- **Dijkstra algoritmus:** Meghatározza a hálózaton belüli legrövidebb útvonalakat.
- **Logolás:** Minden hálózati esemény naplózva van fájlokba.
- **Wireshark kompatibilitás:** A hálózati csomagokat `.pcap` formátumban rögzíti a rendszer.

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

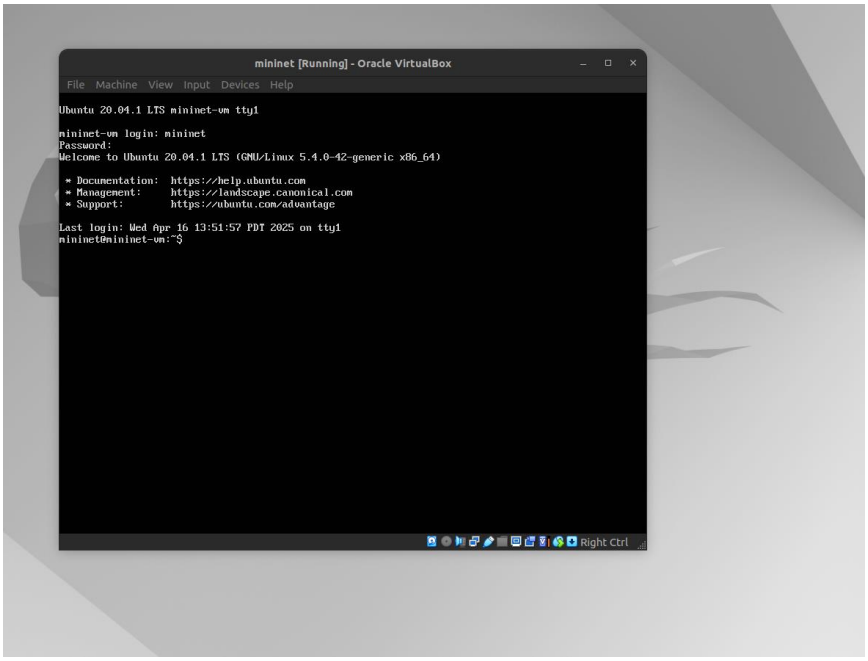
2.4.2. Telepítési lehetőségek

A Mininet virtuális környezet telepítésére, a felhasználó operációs rendszere és felhasználási módszerétől függően.

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. Töltse le a szükséges csomagokat:

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



1. ábra: Mininet VM Virtualbox-ban futtatva

B) Lokális Linux rendszerre letöltés

1. Töltse le a Mininet-et lokálisan a hivatalos oldal alapján.
2. Töltse le a szükséges csomagokat:

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

2.4.3. Csatlakozás és fájlmásolás

SSH-val csatlakozzon a Mininet VM-hez:

```
$ ssh mininet@[IP-cím]
```

Másolja be a projektfájlokat:

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

2.4.4. A program futtatása

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. Ezekenek 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 Wiresharkban: **`[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

2. 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.

3. 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.

3.3. Hasznos fogalmak

3.3.1. Alap fogalmak:

- **OSPF (Open Shortest Path First):** Kapcsolat-állapot alapú útvonalválasztó protokoll, amely az IP hálózatokban a leggyorsabb útvonalakat számolja ki.
- **Kapcsolat-állapot protokoll:** Olyan útvonalválasztó protokoll, amely az egész hálózat topológiájának ismeretében számít útvonalakat.
- **Router:** Olyan hálózati eszköz, amely különböző hálózatok között továbbít adatcsomagokat.
- **Hálózati topológia:** A hálózati eszközök fizikai vagy logikai elrendezése.

3.3.2. Protokoll specifikus fogalmak:

- **Hello üzenet:** Az OSPF által használt üzenettípus, amely a szomszédos routerek felfedezésére és szomszédsági kapcsolat fenntartására szolgál.
- **RID (Router ID):** Az OSPF-ben résztvevő routerek egyedi azonosítója.
- **Area ID:**
- **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.

3.3.3. Technikai fogalmak:

- **Mininet:** Egy hálózatszimulátor 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 és leállítja a routereken futó OSPF folyamatokat.
    c. Monitorozza a hálózati eseményeket.
2. **OSPF modul**
    a. Szimulálja az OSPF Hello protokollt.
    b. Kezeli a szomszédállapotok változásait.
    c. LSDB-t (Link-State Database) épít és karbantart.
3. **Monitoring modul**
    a. A logfájlok és PCAP állományok kezeléséért felelős.
    b. Folyamatosan követi a hálózati kommunikációt.

További elemek:

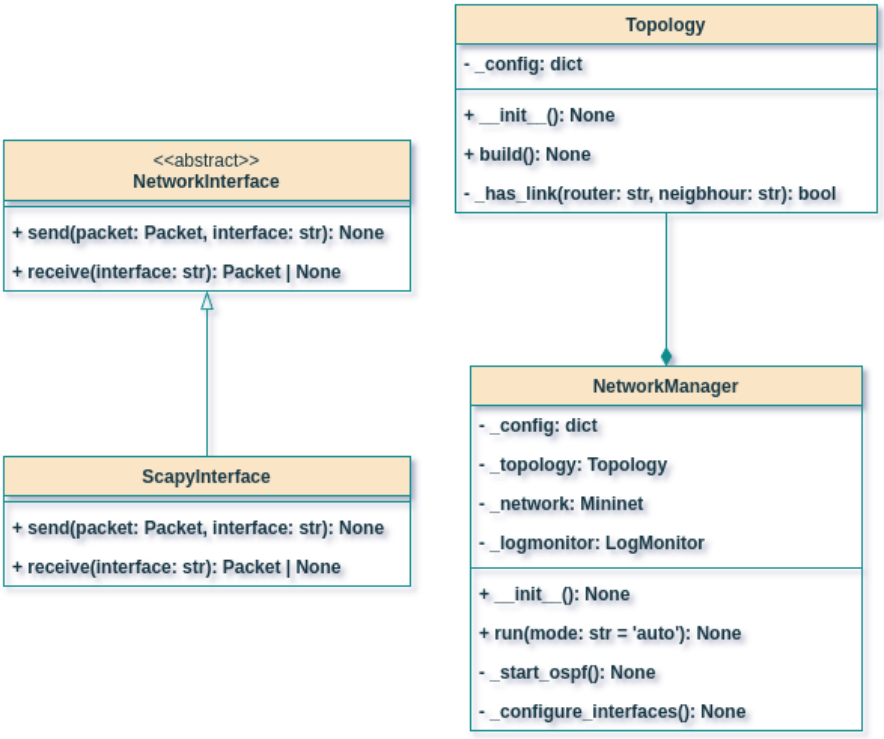
- `common/` könyvtár: Itt tárolom a több osztály és modul által használt segédfüggvényeket, például a YAML konfigurációs fájlok parse-olására használt `get_config()` függvényt.
- `tests/` könyvtár: egységtesztek tárolása.

3.5. A megvalósított modell

Mind a három modul különböző osztályokból épül fel, ezeknek az osztályszintű leírása a következőkben olvasható. Még ennél is részletesebb leírás a kódban, a függvények docstring-jeiben találhatóak meg.

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.



5. 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 ás 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 futásához összes szükséges adatszerkezetét és osztályát.

6. 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 és a legrövidebb ót kiszámolását majd a topológia megjelenítését. Az olvashatóság kedvéért az osztály több logikai szekcióra lett felbontva. Az OSPF folyamatot routere-ken belül is, interfészenként indítom el.

Minden olyan folyamat, ami folyamatos futást vagy más folyamatokkal való párhuzamos futást igényel külön szálon indítottam el. Ilyen például a _listen(), ami a ScapyInterface-t felhasználva egyfolytában figyeli a beérkező csomagokat, a _send_hello(), ami 10 másodpercenként Hello csomagokat multicast-el, az _is_down() ami figyeli, hogy a szomszédoktól érkező utolsó Hello csomag a 40 másodperces Dead Intervalon belül volt-e. És a _state_watch(), ami egyfolytában figyeli a szomszédok állapotát és kezeli azoknal a megváltoztatását. Ezek mellett fut még egy _intf_monitor(), ami figyeli a router interfészeinek az állapotát és kezeli az ennek megváltozásából adódó eseményeket. Az utolsó külön futó folyamat a _process_packet() szál, ami figyeli a _listen() által feltöltött csomag-sort és az ide belekerülő csomagokat típustól függően feldolgozásra küldi.

Ezeknek a folyamatoknak az inditásáért, majd bevárásáért és biztonságos leállásáért az Életciklus kezelés szekció felel. A folyamatok futás közben egy _stop_event nevű threading.Event beállítását nézik és veszik észre, hogy amint tudnak le kell állniuk.

A _listen() és _process_packet() által reprezentált Hálózati csomagkezelés szekció után a Hello csomagok kezelése következik. A router a nem Down állapotban lévő már ismert szomszédait tartalmazó Hello csomagokat küld a multicast címre, majd a beérkező Hello csomagot a 3.7-es pontban említett állapotváltozási feltételek alapján dolgozza fel és tárolja el az információkat a _neighbour_states() Python szótárban. Ebben a szótárban a router minden szomszédjáról tárol egy Neighbour példányt, ami többek között tartalmazza a szomszéd RID-ját és hogy mikor küldött utoljára Hello üzenetet.

A következő LSUpdate csomagok kezelése szekcióban a már Full állapotban történő LSA flood-olás kezelése történik. A router a _generate_router_lsa() függvénnyel legenerálja a saját LSA-ját, ami tartlamazza egy lsitában OSPF_Link objektumként a közötte és a szomszédja közötti kapcsolatot, súllyal együtt. Ezek az LSA csomagok tartlamaznak egy szekvencia számot, amit a szomszéd fogadáskor ellenőriz és ezalapján eldönti, hogy a kapott csomag a legújabb verziójú LSA-e. A router ezután beleteszi az adatbázisába a generált LSAt-t és minden FULL állapotban lévő szomszédjának kiküldi a _flood_lsa() segítségével. Amikor pedig LSUpdate csomag érkezik a router egyesével feldolgozza a csomag LSA listájában lévő LSA-kat. A szekvencia szám alapján eldönti, hogy friss információt kapott-e, ha igen, ezt eltárolja az adatbázisában, lementi a beérkezés időpontját, majd ezt a _flood_lsa()-val továbbterjeszti minden olyan FULL állapotban lévő szomszédjának, aki nem a csomag küldője.

Az utolsó fő logikai szekció a Topológia és legrövidebb út, ami nevéből is adódóan kiszámítja az routerek adatbázisának tartalma alapján a legrövidebb utat, majd ezeket a topológia információkat eltárolja. Ebből ezután a _show_topology() segítségével felrajzol egy egyszerű terminálban is könnyen megjeleníthető topológiát és beállítja a router routing tábláját.

Az algoritmus futása mivel folyamatos, CTRL+C billentyű kombináció megnyomásával ajánlott leállítani. Ezt a program felismerve beállítja a _stop_event() változót és a folyamatoka ahogy ezt észreveszik elkezdenek leállni, ami beletarthat 1-2 másodpercbe.

Az OSPF futása során a router egy eltárolt InfoLogger példány segítségével minden állapotváltozást és hálózati kommunikációt logfájlokba tárol és ezt Monitoring modul a felhasználó számára a fő terminálban majd megjeleníti. Emlett a PcapLogger eltárolt példánya segítségével minden hálózati csomag küldése és fogadása után a program lementi logfájlba a csomagokat későbbi elemzésre.

Figyelem! A különböző logfájlok minden indítás során routerenként törlődnek, ezeknek az időbeni lementésére oda kell figyelni.

7. ábra: OSPF osztálydiagram

3.5.2.2. Class::LSDB

A LSDB a router kapcsolat-állapot adatbázisának egy reprezentációja, ami az LSA-kat típusonként tárolja egy listában. Az osztály metódusai lehetőseget adnak az adatbázisból a csomagok kiszedésére, beletételére és az összes csomag egy listában való visszaadására. Ezek a műveletek elengedhetetlenek az adatbázis karbantarthatósága érdekében.

3.5.2.3. Class::Neighbour

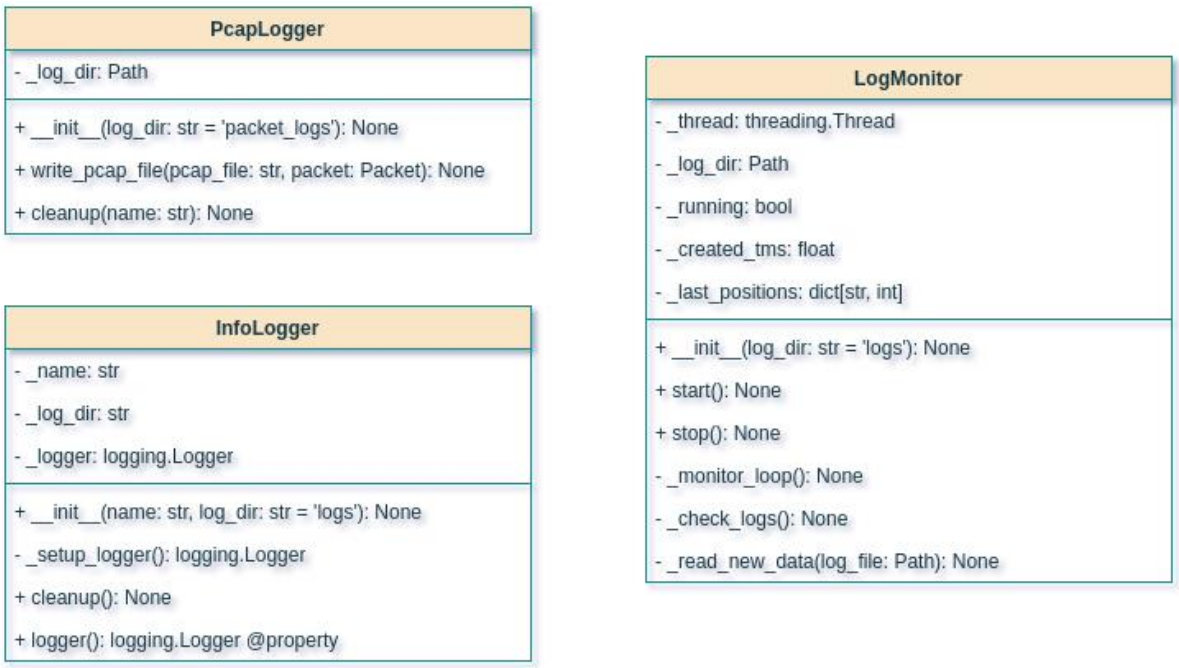
A Neighbour osztály példányait az OSPF osztály használja és a _neighbour_states Python szótárban interfészenként tárolja. Itt tárolja a router a szomszédról szerzett adatait, mint a RID-ja, az IP-címe, MAC-címe, mikor érkezett tőle utoljára Hello csomag és mi az aktuális állapota.

3.5.2.4. Class::State

A State osztály az OSPF és Neighbour osztályok által használt egy felsoroló, ami tartlamazza az összes lehetséges állapotot, amit a szomszéd felvehet az algoritmus futása során.

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.



8. 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, név szerint az adott router nevével elmentve.
- 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 fájlba.
- 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.

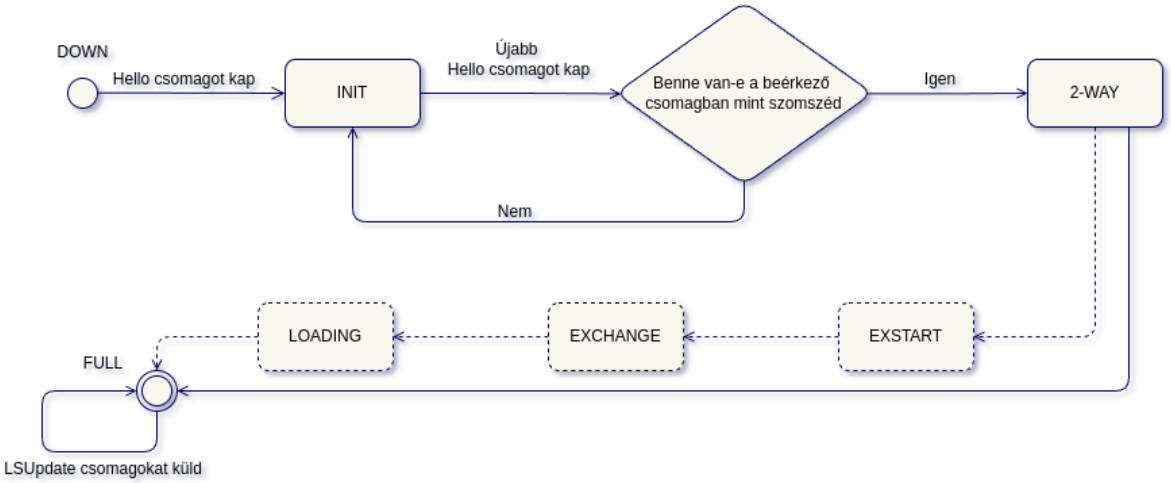
- **`packet_logs/`** könyvtár: Az interfészek közötti hálózati kommunikációt PCAP formátumban tárolja. A fájlok a következő névformátummal jönnek létre: **`[RouterNév]-[InterfészNév].pcap`**. Ezek a fájlok a szimuláció végén másolhatók le a Mininet gépről, és Wireshark-kal részletesen elemezhetők.

  A PCAP fájlok lehetővé teszik a szimulált OSPF protokoll mélyebb szintű, csomagszintű vizsgálatát és a hálózati viselkedés finom részleteinek feltérképezését.

3.7. OSPF szomszéd állapotok (Neighbor States)

A routerek szomszédos interfészei az alábbi állapotokon mennek át:

- **Down:** Kezdeti állapot. Azt jelzi, hogy még nem történt Hello csomag küldése vagy már eltelt Dead Interval-nyi idő és nem érkezett Hello csomag a szomszédtól.
- **Init:** A router Hello csomagot küldött a multicast címre, de még nem kapott választ. A csomagban elküldi azt is milyen szomszédokat ismer.
- **2-Way:** Olyan Hello csomag érkezett, aminek a szomszéd listájában felfedezi magát. Kölcsönösen felismerték egymást a szomszéddal és elkezdik az adatbázis szinkronizációt.
- **ExStart, Exchange, Loading: S**zinkronizációs lépések. Ezek az állapotok a kódban jelenleg csak reprezentatív módon jelennek meg.
- **Full:** teljes adatbázis szinkronizáció, teljes kapcsolat.



8. ábra: OSPF állapotdiagram

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:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.8.2.3. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.8.2.4. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

3.8.2.5. Teszteset:

Bemenet	
Elvárt kimenet	
Tényleges kimenet	

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.

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