ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Gyurcsovics Endre Ákos**

2023 Hallgató törzskönyvi száma: **T/006771/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Gyurcsovics Endre Ákos**
Törzskönyvi száma: T/006771/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Open-Mesh alapú privát VPN szolgáltatás kialakítása
Development of a private VPN service based on Open-Mesh architecture

Intézményi konzulens: Emődi Márk Benjámin
Külső konzulens:

Beadási határidő: 2022. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Felhőszolgáltatási technológiák és IT biztonság specializáció

A feladat

Modern korunk társadalmának szembe kellett néznie egy világjárvánnyal, amiben az egészség megőrzése érdekében rákényszerültek az emberek az otthoni életre. Hogy a munkavégzés zavartalanul működjön, valamint a munkavállalók hozzáférjenek a megfelelő erőforrásokhoz a vállalatoknak VNP szolgálatásokat érdemes használniuk a belső rendszereik biztonságos távoli elérésének biztosítása érdekében.

A szakdolgozó feladata egy olyan rendszer tervezése és fejlesztése, mely képes alkalmazkodni a hálózati változásokhoz, valamint képes alkalmazkodni az otthonokban kialakult folyamatosan változó környezethez. A rendszer tervezése során vegye figyelembe, hogy az architektúra terhelése dinamikusan változhat, az útvonalakat érdek alapján lehessen befolyásolni, valamint olyan megoldást készítsen, mely képes a rendszer skálázhatóságát megfelelően kezelni. Megvalósítás során vizsgálja meg a rendszer erőforrásigényét is. A dolgozat eredménye erre a problémára nyújthat átfogó megoldást Mesh architektúra alapon.

A dolgozatnak tartalmaznia kell:

- a Mesh technológia ismertetését,
- a Mesh hálózati rétegének bemutatását,
- a tesztkörnyezet kialakítását,
- a tesztkörnyezetben kiértékelt eredmények alapján a Mesh architektúra megtervezését,
- a rendszer megvalósítását,
- a rendszer mérési/tesztelési dokumentációját,
- a rendszer továbbfejlesztési lehetőségeinek bemutatását.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 20.22.12.11.....

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Gyurcsovics Endre Ákos [redacted] Nappali

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Open-Mesh alapú privát VPN szolgáltatás kialakítása

Szakdolgozat / Diplomamunka[2] címe angolul:

Development of private VPN service based on Open-Mesh architecture

Intézményi konzulens: Külső konzulens:

Emődi Márk Benjamin

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.11	Feladat pontosítása	Emődi Márk
2.	2022.03.25	Irodalomkutatás megtervezése	Emődi Márk
3.	2022.04.11	Tesztkörnyezet javaslatok átbeszélése	Emődi Márk
4.	2022.04.11	Végleges rendszerterv konzultációja	Emődi Márk

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.11.25

Emődi Márk
.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Gyurcsovics Endre Ákos [redacted] Nappali

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Open-Mesh alapú privát VPN szolgáltatás kialakítása

Szakdolgozat / Diplomamunka[2] címe angolul:

Development of private VPN service based on Open-Mesh architecture

Intézményi konzulens: Külső konzulens:

Emődi Márk Benjamin

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.09.23.	Megvalósítás lépesinek egyeztetése	Emődi Márk
2.	2022.10.07	Erőforrások meghatározása	Emődi Márk
3.	2022.10.28	Kód logikai elemeinek megbeszélése	Emődi Márk
4.	2022.11.25	Szakdolgozat Összegzése	Emődi Márk

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Javasolt érdemjegy: 5

Emődi Márk
.....
intézményi konzulens

Budapest, 2022.11.25

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

1. Absztrakt

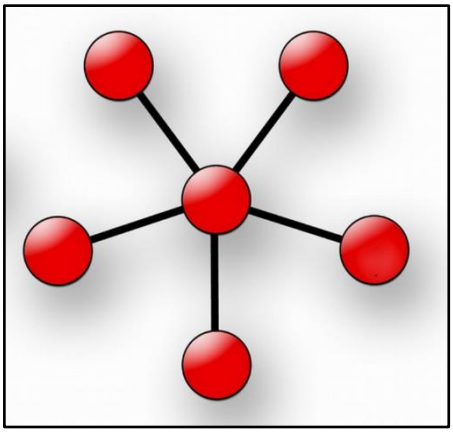
Modern korunk társadalmának szembe kellett néznie egy világjárvánnyal, amiben az egészség megőrzése érdekében rá kényszerültek az emberek az otthoni munkavégzésre. Azonban, hogy a munkavégzés zavartalanul működjön, a vállalatoknak VPN szolgálatásokat kell használniuk a belső rendszereik távoli elérése érdekében. Dolgozatom során erre a helyzetre keresek olyan megoldást, amely képes alkalmazkodni az otthonokban kialakult folyamatosan változó környezethez. Továbbá kezeli a megnövekedett hálózati forgalmat és dinamikusan változtatja az útvonalait. Egyszerűen skálázhatóak a hálózat csomópontjai és mindezek mellett kevésbé erőforrás igényes. Erre a problémára lehet megoldás a Mesh architektúra. Amiben a végpontok erre a topológiára jellemzően, Ad hoc jellegűen csatlakoznak fel, illetve le.

2. Bevezetés

A XXI. századi ember életében, a technológiák fejlődése következtében olyan változások mentek végbe, amik 50 évvel ezelőtt elképzelhetetlenek lettek volna. Legyen szó akár egészségügyről, közlekedésről, számítástechnikáról, számtalan újítást fel tudunk sorolni minden területről, amik könnyebbé tették életünket. Gondolhatunk itt navigációra, egészségünk monitorozására, okos otthonokra egyaránt. Ezeket mind, egy forradalmi technológia fogta össze, az okostelefonok. Az okostelefonok és a közösségi média létrejöttével az információ áramlásnak egy új korszaka kezdődött. Az online térben képesek vagyunk kommunikálni ismerőseinkkel, barátainkkal, legyenek akár a világ bármely pontján. Mindent megoszthatunk az üzenőfalainkon egy pillanat alatt. Ezzel kialakítva egy online lenyomatot magunkról.

Amennyivel leegyszerűsítette a közösségi média a kapcsolatainkat, annyival éreztük szükségtelennek a fizikai találkozásokat. Amikor beköszöntött a Covid-19 világjárvány, újra kellett értékelnünk mi is igazán fontos az életünkben. Az egészségünk megőrzése érdekében, új alternatívákat kellett alkalmaznunk életünk különböző területein. Próbáltuk csökkenteni a fizikai kontaktussal járó tevékenységet, hogy ezzel is kizárjuk a vírus fertőződés kockázatát. Például a boltba járás helyett, egyszerű online ételkiszállítást választottunk, az iskolákba pedig átálltak a teljeskörű online oktatásra. A karantén ideje alatt, hogy minimalizáljuk a vírus terjedésének lehetőséget, elkerültük az irodahelységeket. Azonban a gazdaság nem állhatott meg, így át kellett állnunk az otthoni munkavégzésre. [1]

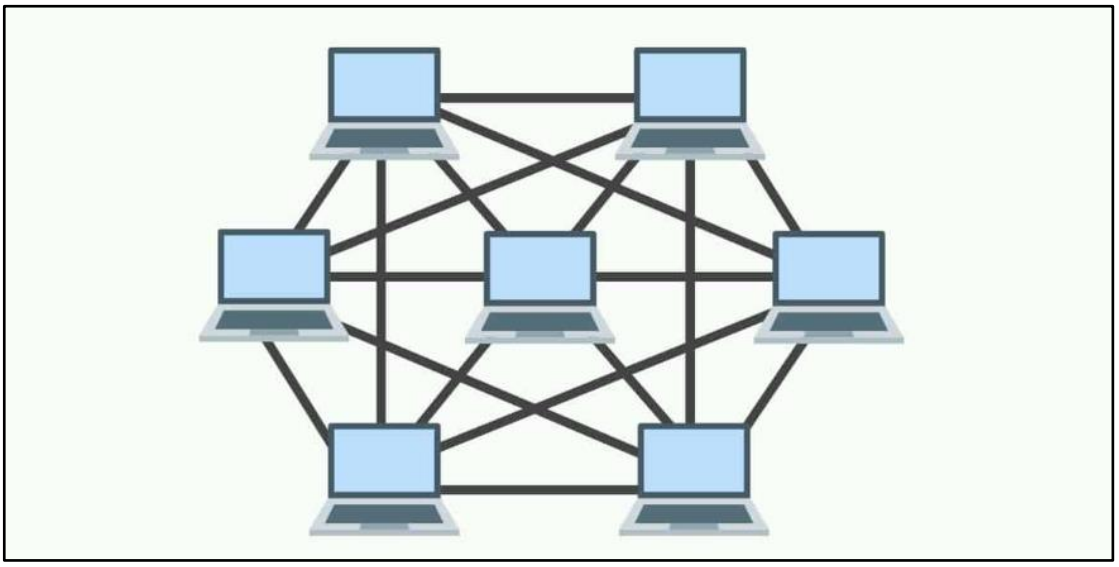
Ahhoz, hogy egy jól funkcionáló teljes értékű munkakörnyezetet ki tudjunk alakítani, el kell érnünk a vállalat által használt belső rendszereket. Többféle VPN rendszer közül választhatunk, de a rendszernek kezelnie kell a hirtelen kapcsolat vesztés esetén kialakult helyzeteket az otthoni internet megbízhatatlansága miatt. Alkalmasnak kell lennie a nagy teherbírásra, mivel kiszámíthatatlan, hogy mikor hányan akarják használni a rendszert. A skálázhatóság pedig elengedhetetlen, hogy költség hatékony legyen a rendszer. A fentebb említett nehézségekre egy lehetséges megoldást kínál az OpenVPN Community verziója, mivel képes több ezer kapcsolatot egyszerre kezelni, rugalmas megoldást nyújt kapcsolat vesztésre az SSL technológia miatt és mindemellett ingyenes. A hálózat topológiáját tekintve az elsődleges szempont a megbízhatóság. Ezért megvizsgáltam több topológiát,



2.1. ábra csomópontok összekapcsolása csillag topológiában [2]

amiből szeretnék kiemelni kettőt, amik ígéretesek voltak. Elsőként a csillagtopológiát vizsgáltam meg, ahol minden csomópont egy központi szerveren kommunikál, azon keresztül lépnek kapcsolatba egymással a csomópontok. Egy viszonylag egyszerű, költséghatékony topológiáról beszélünk, emellett, ha kapcsolat vesztés lépne fel egy kliens csomópont esetén, csak egy gép esne ki a rendszerből. Ugyanakkor nagy hátránya, hogy amennyiben a központi gép meghibásodik az egész rendszer megbénulhat, ezzel tulajdonképpen működés képtelenné téve a hálózatot. Ez sajnos kritikus lenne a rendszerünk szempontjából igy nem lenne ideális választás. [2]

A második megvizsgált topológia a mesh topológia, amit szoktak Ad-hoc topológiának is nevezni. Egy kicsit komplexebb megoldásról beszélünk a hálózat tervezését illetően. A mesh topológia könnyen kiküszöböli a csomóponti hibákat a redundáns szerkezeti miatt. A pont-pont kapcsolat jellegzetessége, hogy minden gép kapcsolatban van egymással. Az architektúra erőssége, hogy nem érzékeny időszakos hálózati kiesésre, a rendszer nagyrésze használható marad. Mint ahogy azt látjuk egy rendkívül megbízható topológiáról beszélünk, ami megoldás a problémánkra.



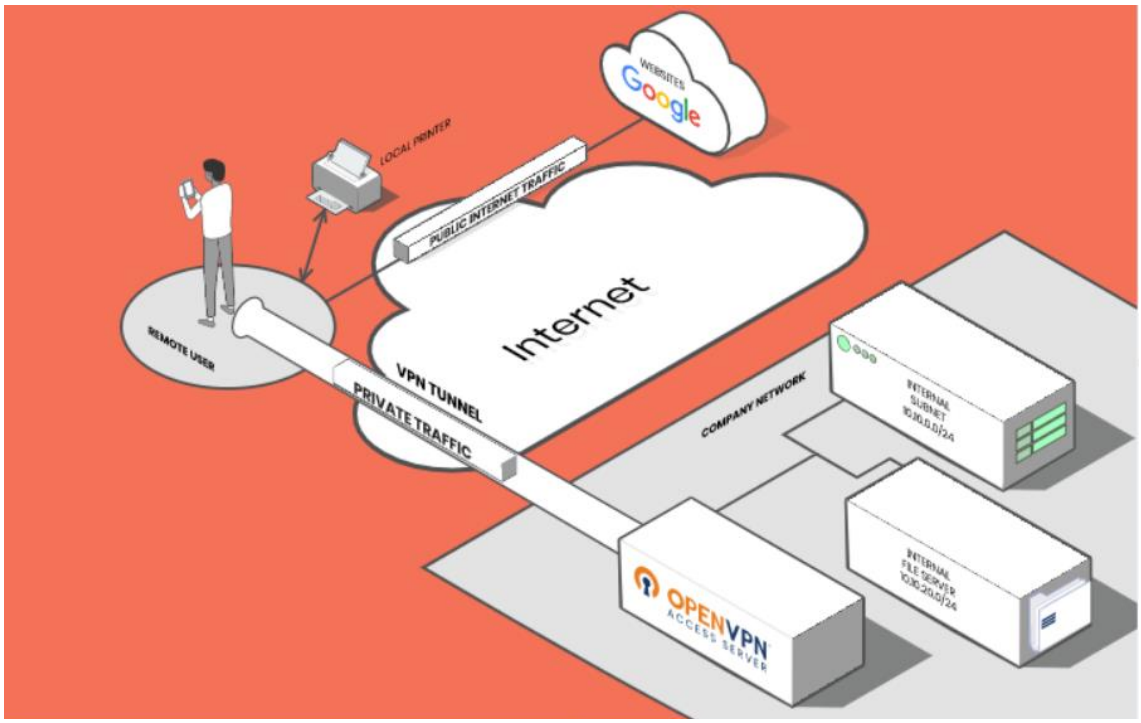
2.2. ábra Mesh topológia szemléltetése [14]

Ez lett végül szakdolgozatom alapja, mert számos lehetőség van benne, a komplexitás ellenére. A szakdolgozatom célja, hogy csökkentsem a hálózat kialakításának komplexitást a felhasználó szempontjából. A szakdolgozatom során egy olyan integrációt tervezek készíteni, amely képes egy újonnan csatlakozó végeszközt beléptetni a hálózatba, ami ellátja azt minden szükséges konfigurációval annak érdekében, hogy a hálózat zavartalanul működjön az új eszköz bevonásával. Mindezt szeretném, hogy egy grafikus felhasználói interfész segítségével lehessen vezérelni, megkönnyítve ezzel a hálózat konfigurálását.

3. OpenVPN

Az OpenVPN community verziója egy teljes körű, nyílt forráskódú SSL VPN megoldás, amely a rendkívül sok oldalú, konfigurációk széles skáláját támogatja. Az OpenVPN a nyíltforráskódú miatt rohamos mértékben fejlődik. Ennek eredménye, hogy rövid időn beül a Linux kernelek is alapértelmezetten tartalmazni fogják ezért remek választottam szakdolgozatomhoz. Alkalmazható távoli hozzáférésre, könnyen használható helyközi VPN-nek kialakításához. Képes vállalati szintű távoli hozzáférési megoldásokat terheléselosztással koordinálni, illetve rendelkezik beépített hibaelhárítással és egyénre szabott hozzáférés-szabályozással. Az OpenVPN alap gondolata az, hogy a komplexitás fordítottan arányos a biztonsággal, minél könnyebben átlátható egy rendszer annál egyszerűbben kezelhető. Ezért egy remek választás mert költséghatékony és könnyen kezelhető alternatívát kínál más VPN-technológiákkal szemben. [3]

Az OpenVPN biztonsági modellje az SSL-en tanúsítványokon alapszik, úgy, ahogy interneten keresztüli biztonságos kommunikáció is zajlik. Az OpenVPN az SSL/TLS protokoll segítségével valósítja meg az OSI 2. illetve 3. rétegű biztonságos hálózati kapcsolatot. Számos ügyfél-hitelesítési módszert támogat az OpenVPN többek között tanúsítványon, illetve 2-faktoros hitelesítésen alapuló technológiákat. Ezek mellett lehetővé teszi a felhasználó vagy csoport specifikus hozzáférési szabályozást. Amiket a virtuális interfészein létrehozott tűzfalszabályozások segítségével visz véghez. [3]



3.1. ábra OpenVPN infografika [17]

3.1. OpenVPN használata

A VPN-ek fontossága – ahogy már említésre került a bevezető fejezetben – elengedhetetlen a jelen korunk emberének munkája során. Egy jól működő VPN

szolgáltatást a felhasználó észre sem veszi és zökkenőmentesen tudja használni a belső rendszereket. Ezért szeretném kielemelni fontosabb tulajdonságait az OpenVPN-nek, amelyek lényegesek lehetnek amikor a szolgáltatást szeretnénk kialakítani.

- Az OpenVPN képes terhelés elosztásra, illetve nagy mértékű skálázhatóságra, konfigurálhatunk VPN szervereket, amelyek képesek lesznek dinamikusan akár több ezer beérkező kliens kapcsolatát kezelni egyidőben minden probléma nélkül.
- Nagy segítség lehet privát hálózatunk kialakításában mivel legtöbb alhálózati IP címmel vagy virtuális Ethernet adapterrel képes működni mindösszesen csak egy TCP vagy UDP portra van szükségünk.
- A nagy hálózati forgalom kezelésének érdekében az OpenVPN valós idejű adaptív link tömörítő és forgalom alakító technológiákat használ. Továbbá menedzseli a kapcsolatok sávszélességének kihasználtságát.
- Az OpenVPN kihasználja az legtöbb SSL technológia által nyújtott előnyöket úgy, mint a hitelesítési és tanúsítási funkciókat, amiket alkalmazhatunk privát hálózatunk védelmének kialakításában.
- Választhatunk a statikus kulcson alapuló hagyományos titkosítás vagy tanúsítványalapú nyilvános kulcsú titkosítás között.
- Az OpenVPN képes kialakítani Ethernet a bridgeket a virtuális Tap interfészei segítségével, amit a későbbiekben a teszt környezetben használni is fogunk. [3]

3.2. Az OpenVPN előnyei más rendszerekkel szemben

Az OpenVPN fő erősségei közé tartozik a platformok közötti kompatibilitás, a kiváló stabilitás és megbízhatóság. Továbbá alkalmas több száz vagy több ezer kliensre való skálázhatóságra, emellett viszonylag egyszerű telepítése. Mindemellett támogatja a dinamikus IP-címeket és a NAT technológiát. Az OpenVPN támogatja az IPv6 címzést, ami rendkívül fontos egy korszerű rendszer kialakításában. Emellett rendelkezik az OpenSSL vagy a PolarSSL protokollok támogatásával.

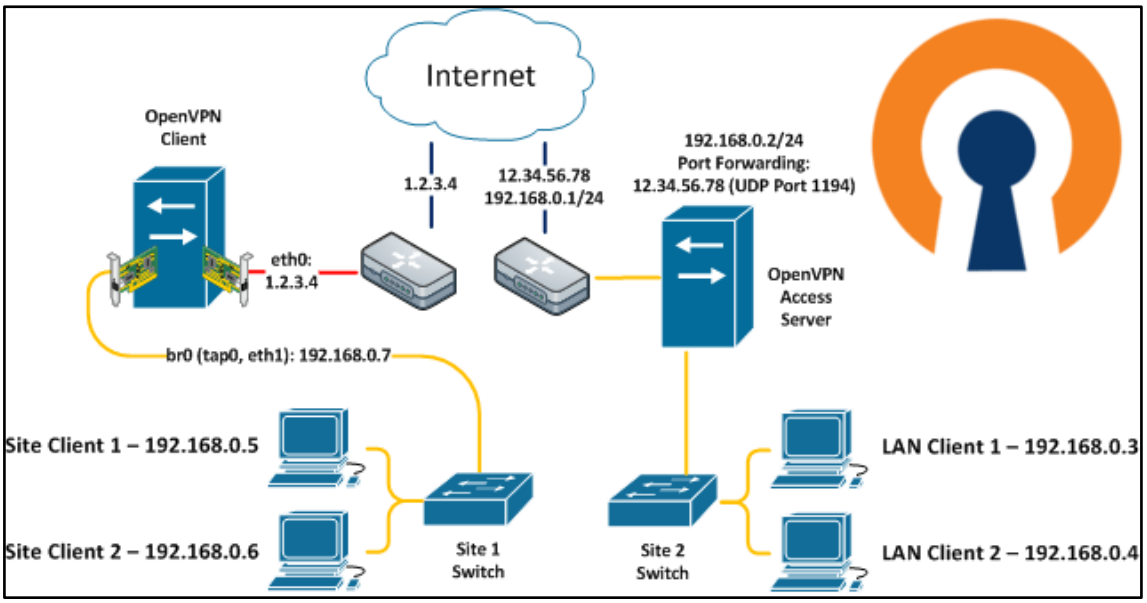
Az OpenVPN-t szigorúan úgy tervezték és tesztelték, hogy megbízhatóan működjön megbízhatatlan hálózatokon is, ami roppant fontos a szempont volt a technológia kiválasztásakor. Az OpenVPN egyik fő tervezési célja, hogy a normál működés és a hibaelhárítás szempontjából is, hogy ugyanolyan tulajdonságokkal rendelkezzen, mint az alapul szolgáló IP-réteg, amelyen keresztül kapcsolatot képez. Ez azt jelenti, hogy ha az IP-réteg 5 percre leáll, amikor újra működésbe lép, a csatorna forgalom azonnal folytatódik, még akkor is, ha a kiesés megzavarta az erre az időre tervezett dinamikus kulcscserét.

Az OpenVPN egy bővíthető VPN keretrendszert biztosít, például lehetőséget ad arra, hogy testreszabott telepítőcsomagot terjesszünk az klienseknek, vagy hogy alternatív hitelesítési módszereket támogasson az OpenVPN modul interfészén keresztül.

Windows alatt az OpenVPN képes tanúsítványokat és privát kulcsokat kezelni a Windows Crypto API-t támogató intelligens kártyákról, de én a szakdolgozatom során Linux alapú (Debian alapú disztribúció) rendszereket használtam.

Az OpenVPN egy olyan ipari szintű biztonsági modellt használ, amelyet a passzív és aktív támadások ellen is meg állja a helyét, biztonsági modellje az SSL/TLS használatán alapul. A munkamenet hitelesítéshez az IPSec ESP protokollt használja.

Az OpenVPN nagy előnye, hogy könnyen telepíthető. Általában egyetlen létrehozható és konfigurálható egy kapcsolat. Ugyanakkor lehetőségünk van definiálni előre virtuális interfészeket, amiket a rendszer automatikusan felismer.



3.2. ábra OpenVpn lehetséges megvalósítás [9]

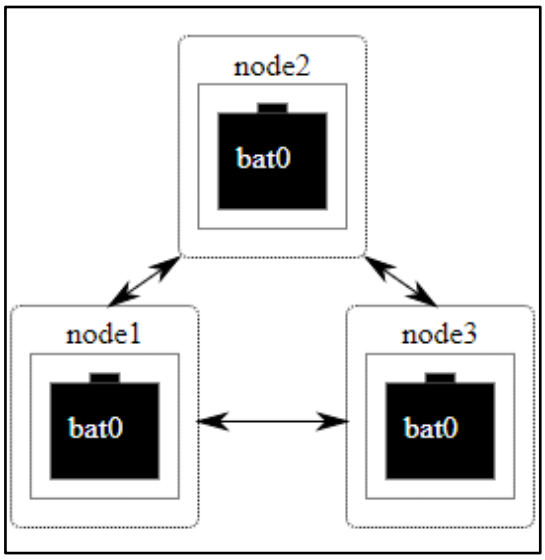
Az OpenVPN erősen moduláris felépítésű. Az összes titkosítást az SSL könyvtár kezeli, az összes IP-csatornát létrehozó funkciót pedig a TUN/TAP virtuális hálózati meghajtó biztosítja. Ennek a modularitásnak az előnye abban mutatkozik meg, hogy az OpenVPN könnyedén szinkronizálható az SSL-könyvtár új verziójával, és azonnal hozzáférhetünk egy új verzióban bevezetett funkciókhoz. Ugyanígy az OpenVPN sokoldalúsága lehetővé teszi az egyszerű portolást bármely olyan operációs rendszerre, amely tartalmaz TUN/TAP virtuális hálózati meghajtót.

4. B.A.T.M.A.N Advanced protokoll

4.1. Batman-adv Működése

A Linux kernelben található útválasztási protokollok a B.A.T.M.A.N protokoll, aminek az egyik implementációja a batman-advanced (későbbiekben batman-adv). A legtöbb vezeték nélküli útvonalválasztási megoldás az OSI rendszer harmadik osztályában található amíg a batman-adv a második osztálynál nem megy feljebb. A csomópontok UDP csomagok küldésével cserélnek útválasztási információkat, majd kernel routing táblájának módosítása után hozzák meg a szükséges útvonal választási döntést. A batman-adv teljes egészében az OSI 2. rétegén működik - nem csak az útválasztási információkat szállítja nyers Ethernet-keretekkel, hanem az adatforgalmat is a batman-adv kezeli. Minden forgalmat bekeretez és közvetít a célállomás felé. Minden csomópont lokális kapcsolatként tűnik fel, emiatt egyik hálózatban résztvevő eszköz sem tudja a hálózat teljes topológiáját és ezért nem befolyásolják őket a hálózatban történő változások. A legegyszerűbben úgy képzelhetjük el a hálózatot, mint egy, mint egy elosztott switchet.

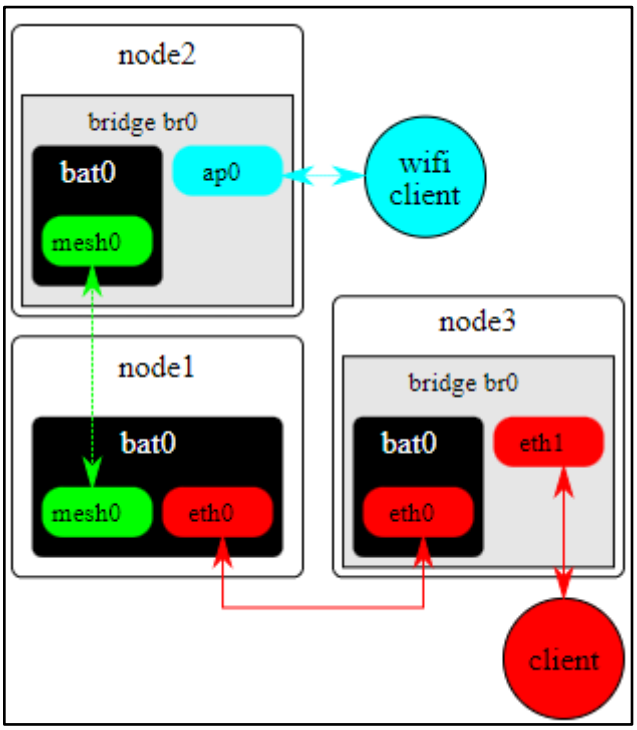
Minden csomóponton a batman-adv egy példánya fut. Ezeken az egységeken találhatóm



4.1. ábra B.A.T.M.A.N csomópontok bat0 interfésszel [4]

egy virtuális hálózati interfész, amit legtöbbször bat0 szoktak nevezni. Ezek az interfészek a rendszer úgy ismeri fel őket, mint egy switch port ami hozzáférést biztosít a többi elosztott switchhez. A hálózat különböző csomópontjai közötti kommunikációhoz a protokoll a Ethernet-kompatibilis hálózati interfészekre támaszkodik, amelyeket a bat0-hoz csatolnak. Egy példán keresztül szemléltetve ezt:

Ahogyan a 5.2. képen is látható a 3 csomópontunk képes kommunikálni a 2 csomóponttal. még akkor is, ha a node2-vel való "mesh0" interfészen keresztül a wifi kapcsolat túl gyenge lenne az adatátvitelhez. Ebben az esetben az egyes csomópont forgalom továbbításra használjuk.



4.2. ábra B.A.T.M.A.N topológia szemléltetése [4]

Az adatok továbbítása Ethernet "eth0" interfészen keresztül történik. A node1 pedig a "mesh0" interfészen keresztül beszélhet a node2-vel. A batman-adv mindezeket a dolgokat belsőleg kezeli, és a felsőbb rétegek egyszerűen használhatják a bat0-t, mint bármely más hálózati interfészt. Ez azt is jelenti, hogy bármilyen más interfész csatlakoztatható a bat0 interfészre. Egy másik Ethernet interfész ("eth1" ebben a példában) használható arra, hogy egy B.A.T.M.A.N. hálózaton kívüli kliensnek kapcsolatot biztosítson a node1, node2, node3 vagy más, B.A.T.M.A.N. hálózaton kívüli egységhez. [4]

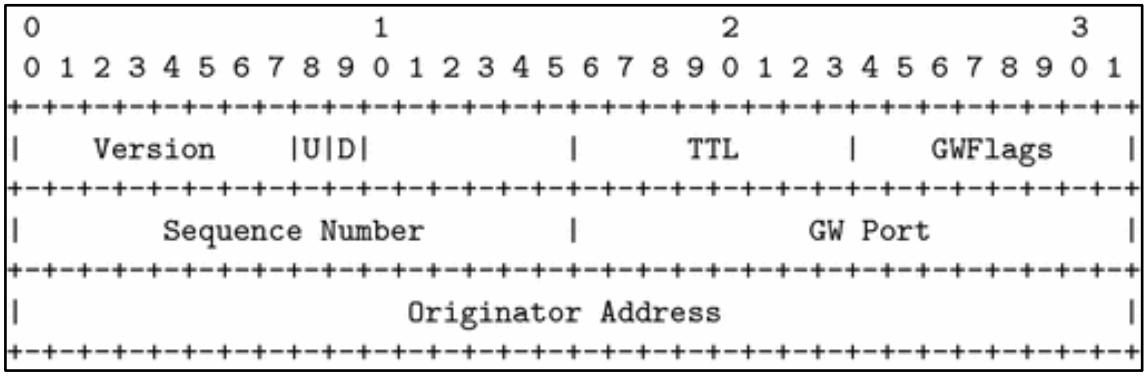
4.2. Az algoritmus

B.A.T.M.A.N algoritmus megközelítése az, hogy a háló csomópontjai közötti legjobb végponttól végpontig tartó útvonalakra vonatkozó tudást szétosztja az összes résztvevő csomópont között. Minden csomópont csak a legjobb következő ugrásról szóló információt észleli és tárolja az összes többi csomópont felé. Az útvonalválasztás egy metrika szerint történik, ahol ha egyik interfészére csomag érkezik megnöveli a metrika értékékét és automatikusan másikat választ. hogy Ezáltal szükségtelenné válik a helyi topológia változásokról szóló globális tudás szükségessége. Ezenkívül az eseményalapú, de időtlen (időtlen abban az értelemben, hogy a B.A.T.M.A.N. soha nem ütemezi a topológiai információkat az útválasztási döntések optimalizálásához) elárasztási

mechanizmus megakadályozza az ellentmondó topológiai információk felhalmozódását (ami az útválasztási hurkok létezésének szokásos oka), és korlátozza a hálót elárasztó topológiai üzenetek mennyiségét. Az algoritmust olyan hálózatok kezelésére tervezték, amelyek megbízhatatlan kapcsolatokon alapulnak. [5]

Minden csomópont broadcast-üzeneteket küld (amiket OGM üzeneteknek nevezünk) hogy a szomszédos csomópontokat tájékoztassa létezéséről. Ezek a szomszédok az OGM-eket meghatározott szabályok szerint újra sugározzák, hogy tájékoztassák szomszédiakat az üzenet eredeti kezdeményezőjének létezéséről, és így folytatódik a folyamat, amíg a teljes hálózat a megfelelő ismeretekhez jut. Így a hálózatot elárasztják a kezdeményező üzenetek. Az OGM-ek kicsik, a tipikus nyers csomagméret 52 bájt, beleértve az IP és UDP felületet is. Az OGM-ek tartalmazni kell minimum a küldő címét, a csomagot továbbító csomópont címét, egy TTL-t és egy sorszámot.

Azok az OGM-ek üzenetek, amelyek olyan útvonalon haladnak, ahol a vezeték nélküli kapcsolat minősége gyenge vagy telített, csomagvesztést vagy késést tapasztalhatunk a hálózaton való áthaladásuk során. Ezért a kevésbé terhelt útvonalakon haladó OGM-ek gyorsabban és megbízhatóbban terjednek a hálózaton. Annak érdekében, hogy meghatározható legyen egy OGM-ről hogy többször érkezett e meg tartalmaz egy sorszámot ami a küldő csomópont határoz meg. Minden csomópont legfeljebb egyszer sugározza újra a hozzá beérkező OGM üzeneteket. [6]



4.3. ábra OGM fejléc tartalma [5]

5. Tesztkörnyezet kialakítása

5.1. Tesztkörnyezet felépítése

A tesz környezet kialakítása során egy olyan hálózatot szeretnék megvalósítani, amiben szimulálni tudom a mesh hálózatokra jellemző tulajdonságokat. Vizsgálni szeretném, hogy a hirtelen csomópont meghibásodásra, hogy reagál a rendszer, hogy képes-e helyettesíteni a megszakadt csatornákat. Továbbá lényeges szempont, hogy mivel a mesh hálózattokat alapból is komplikált architektúrát alkotnak, törekedek az egyszerű kiépítésre és kezdetben egyszerű hálózati topológia kiépítésére. Fontos, hogy könnyen kezelhető legyen a hálózat, de szemléltetni tudja a tőle elvártakat.

A tesztrendszer lépésről-lépésre akartam felépíteni és úgy beleépíteni egyre több technológiát. Először a batman-adv útválasztási technikát akartam a megvizsgálni és annak működését kielemezni. Ehhez egyszerű Linux alapú gépeket akartam használni tekintve, hogy alapból már a kernelbe be van építve ez a modul, de ezt még a későbbiekben részletezni fogom. Így kiválasztottam a Linux családból a Debian 11-es verzióját. Az egyszerűség és könnyen kezelhetőség miatt virtuális gépeket fogok futtatni egy KVM host gépen. A gépeket kiépítése során felhasználtam egy olyan eszközt, amely a virtuális gépek menedzselését valósítja meg. A rendszer python nyelven íródott és azon keresztül futtat bash scripteket a gépeken. Ebben a rendszerben definiáltam az egyes virtuális gépek tulajdonságait.

Példaként bemutatom a „670-BatMat-000” gép specifikációját. A script lehetővé teszi, hogy előre megadhassuk a gépeknek a virtuális hálózati interfészeit. Jelen esetben ahogy a képen is látható 3 darab lesz létrehozva úgy mint: *ens3, ens4, ens5*. Az interfészeknek megadjuk az IP címét, illetve alhálózati maszkját.

Például az IP *ens3* tekintve: *10.96.1.48* illetve a netmaszk 24, ezen keresztül kommunikál a gép az internettel. Továbbá láthatjuk, hogy rendelkezik a gép még 2 interfésszel. Ez a két virtuális interfész szolgálja majd a lokális kapcsolathoz az alapokat. Az egyszerűség miatt még nem OpenVPN csatornákat konfigurálunk, hanem egy másik alternatívát választottam ki, a Linux bridgekkel.

A Linux Bridge teljes mértékben úgy működik, mint egy hálózati switchet. A beérkező csomagokat továbbítja a hozzá csatlakoztatott interfészek között, mindezt a 2. osztály szintjén teszi. Támogatja az feszítőfa (Spanning Tree Protocol, STP) protokollt ezzel elkerülve a hálózati hurkokat. Mindezek mellett roppant egyszerű a beállítása, így jelentős módon egyszerűsíteni tudjuk a tesztkörnyezetünket.

Ahogy a képen is látjuk az *ens4*, illetve az *ens5* virtuális hálózati interfész egy-egy hálózati bridgere van hozzá csatlakoztatva úgy mint: *brBatMan000* és *brBatMan001*. Ezeken a bridgeken keresztül fog majd a kommunikáció végbe menni a gépek között. Tekintve, hogy a bridgek a 2. osztályon folyattatják a csomag továbbítás így az IP címükből kerül generálásra a MAC címük.

Továbbá az interfészeken definiáljuk a DNS szervereket, amik majd a cím fordításokat végzik. A hálózati interfészeken kívül még megadjuk a virtuális gépünknek, hogy mennyi memóriát, illetve mennyi számítási kapacitást használhatnak fel a virtualizáció során. Ehhez hasonlóan határoztuk meg a többi gép beállításait is.

```
Vm(vm_type=VmType.midland_obudaGate_normal, hostname="670-BatMan-000", domain="midland.hu",
    distribution=machineOs.getLatestDistibutionByFamily("debian"),
    nics=[
        Nic(nic_manager=NetworkManager.systemdNetworking,
            nic_type=NicType.dhcp,
            bridge="br96OfficeOG",
            device_name="ens3",
            ipv4interface=ipa.IPv4Interface(u"10.96.1.48/24"),
            subnet_router_ip="10.96.1.1",
            dns_servers_ipv4=[ipa.IPv4Address(u"8.8.8.8"), ipa.IPv4Address(u"62.77.203.10")],  # google, invitech
            generate_mac_from_ip=True, set_dhcp=True
           ),
        Nic(nic_manager=NetworkManager.systemdNetworking,
            nic_type=NicType.static,
            bridge="brBatMan000",
            device_name="ens4",
            ipv4interface=ipa.IPv4Interface(u"10.32.1.1/24"),
            subnet_router_ip="10.32.1.1",
            dns_servers_ipv4=[ipa.IPv4Address(u"8.8.8.8"), ipa.IPv4Address(u"62.77.203.10")],  # google, invitech
            generate_mac_from_ip=True
            ),
        Nic(nic_manager=NetworkManager.systemdNetworking,
            nic_type=NicType.static,
            bridge="brBatMan001",
            device_name="ens5",
            ipv4interface=ipa.IPv4Interface(u"10.33.1.1/24"),
            subnet_router_ip="10.33.1.1",
            dns_servers_ipv4=[ipa.IPv4Address(u"8.8.8.8"), ipa.IPv4Address(u"62.77.203.10")],  # google, invitech
            generate_mac_from_ip=True
            )
    ],
    ram="1024", vcpu="4",
    disks=_set_midland_obudagate_disks(
        root_size="3G",
        swap_size="512M",
        boot_size="256M"
    )
),
```

5.1. ábra Virtuális gép generálása python scriptből

Ahhoz, hogy a szimuláló környezet szemléltetni tudja az esetleges csomópont hibát, kellő átgondolás után arra jutottam, hogy 4 darab virtuális gépre lesz szükségünk. Ez azt jelenti, hogy 4 darab bridge interfészt fogunk létrehozni. A 4 virtuális gép interfészeinek leírása az alábbi táblázatban tekinthető meg:

Virtuális gép	NIC IP címe	NIC MAC Címe	Bridge
670-BatMan-000	10.32.1.1	0a:20:01:01:49:50	brBatMan.000
	10.35.1.2	0a:23:01:02:49:50	brBatMan003
671-BatMan-001	10.32.1.2	0a:20:01:02:49:50	brBatMan000
	10.33.1.1	0a:21:01:01:49:50	brBatMan001
672-BatMan-002	10.33.1.2	0a:21:01:02:49:50	brBatMan001
	10.34.1.1	0a:22:01:01:49:50	brBatMan002
673-BatMan.-003	10.34.1.2	0a:22:01:02:49:50	brBatMan002
	10.35.1.1	0a:23:01:01:49:50	brBatMan003

1. táblázat virtuális gépek adatai

Az interfészek MAC címe nem csak a bridgeknek alapvető a kommunikációban, hozzá hasonlóan a batman-adv is ennek segítségével kommunikál. Így amikor majd egy traceroute paranccsal teszteljük a későbbiekben az útvonalakat, akkor az egyes interfészek MAC címeit fogjuk eredményül kapni.

A gépen installálása után létre hoztuk az átjárókat a megfelelő névvel. Az egyszerűség kedvéért a gépekre a nevüknek az utolsó számjegyével fogok hivatkozni. Topológia tesztelése érdekében pingeltem a 0 gépről az 1 eszközt, ami sikeres volt. A következő lépés a gépeken fel konfigurálni a batman-adv protokollt.

5.2. Batman-adv konfigurálása

Elsőként a gépeken ki adjuk a ***apt-get update*** parancsot hogy frissítsük a csomagokat. Majd le kell töltetünk mind a négy gépre a batman-adv csomagot, amit egyszerűen az ***apt-get install batctl*** paranccsal tehetünk meg. Miután felkerültek be kell lépnünk a */etc/modules-load.d/* könyvtárba, ahol módosítanunk kell a betöltendő modulok listáját, a *module.conf* fájlt. Ebben a fájlban szereplő modulokat a system-d automatikusan betölti majd a rendszer indulásánál, tehát írjuk bele *batman-adv* modult.

Ezután rendszerünkben létre kell hoznunk a batman-adv protokollhoz szükséges bat0 interfészt. Amit a system-d esetében network kiterjesztésű fájlok segítségével tehetünk meg. Tehát lépjünk be a system-d könyvtárába */etc/systemd/network,* majd hozzuk létre a hálózati konfigurációs fájlunkat a következőképpen.

```
[Match]
Name=bat0

[Network]
Address=192.168.123.X/24 ##
Broadcast=192.168.123.255
Gateway=192.168.123.1

IPMasquerade=no

DNS=8.8.8.8
DNS=62.77.203.10

generate address
```

5.2. ábra bat0.network konfigurációs fájl tartalma

Ahogy a képen látjuk különböző szekciókra van osztva fájl, minden szekció rendelkezik egy attribútummal. Elsőként a [Match] részben megadhatjuk, hogy mi legyen az interfészünk neve. Ahhoz, hogy a batman-adv protokoll a későbbiekben felismerje az interfészt *bat* előtaggal kell elneveznünk utána pedig a 0 sorszám a hálózatunk azonosítóját jelenti. Amennyiben több mesh hálózatot szeretnénk kialakítani akkor az utótag számát növelve tehetjük ezt meg. Ez után következik a [Network] szekció, ahol megadhatjuk az interfész IP címét és hozzá a megfelelő netmaszkot, broadcast illetve az alapértelmezett átjáróját. Amennyiben megadjuk ezt a mezőt a rendszer automatikusan felveszi az alapértelmezett útvonalak közé. Ugyan a B.A.T.M.A.N. 2 osztály szintjén valósítja meg az útvonalválasztást szükségünk van IP címekre a pingek, traceroutok érdekében. A tesztkörnyezetünkben a virtuális gépek bat hálózatának 192.168.123.0 IP tartományt választottam 24 maszkkal. Végül megadjuk a DNS szerverek címét és kiadjuk a generate addresst ami generál egy MAC címet a virtuális interfészhez.

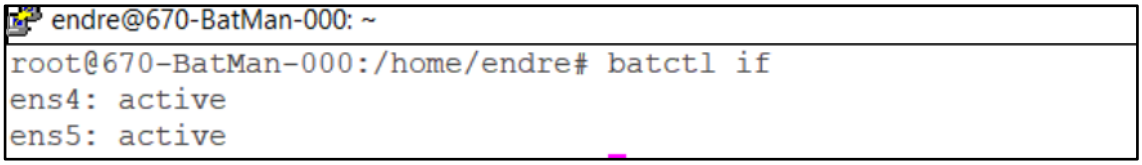
A konfigurációs fájl elkészítése után ahhoz, hogy a system-d elkészítse az interfészt szükséges annak újraindítása, amit a ***systemctl restart systemd-networkd*** paranccsal tehetünk meg. Ezután már látható lesz az interfészek között, amit megnézhetünk az ***IP a*** sorral:

```
root@670-BatMan-000:/etc/systemd/network# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 0a:60:01:30:49:50 brd ff:ff:ff:ff:ff:ff
    inet 10.96.1.48/24 brd 10.96.1.255 scope global dynamic ens3
       valid_lft 597sec preferred_lft 597sec
    inet6 fe80::860:1ff:fe30:4950/64 scope link
       valid_lft forever preferred_lft forever
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bat0 state UP group default qlen 1000
    link/ether 0a:20:01:01:49:50 brd ff:ff:ff:ff:ff:ff
    inet 10.32.1.1/24 brd 10.32.1.255 scope global ens4
       valid_lft forever preferred_lft forever
    inet6 fe80::820:1ff:fe01:4950/64 scope link
       valid_lft forever preferred_lft forever
4: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bat0 state UP group default qlen 1000
    link/ether 0a:23:01:02:49:50 brd ff:ff:ff:ff:ff:ff
    inet 10.35.1.2/24 brd 10.35.1.255 scope global ens5
       valid_lft forever preferred_lft forever
    inet6 fe80::823:1ff:fe02:4950/64 scope link
       valid_lft forever preferred_lft forever
5: bat0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 0a:20:01:01:49:50 brd ff:ff:ff:ff:ff:ff
    inet 192.168.123.2/24 brd 192.168.123.255 scope global bat0
       valid_lft forever preferred_lft forever
    inet6 fe80::820:1ff:fe01:4950/64 scope link
       valid_lft forever preferred_lft forever
```

5.3. ábra 670-BatMan-000 gép hálózati intefészei

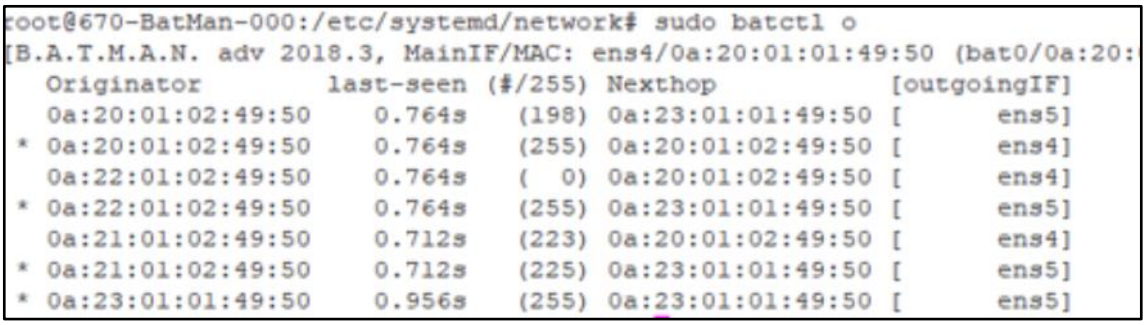
A következő lépésünk az *ens* interfészek hozzáadása lesz a B.A.T.M.A.N. hálózathoz. Amit az alábbi módon tehetünk meg: ***batctl if add ens4*** illetve ***batctl if add ens5***

Amennyiben felkapcsoltuk az interfészeket a ***batctl if*** parancsot beírja az alábbi állapotot kell lássuk: [7]



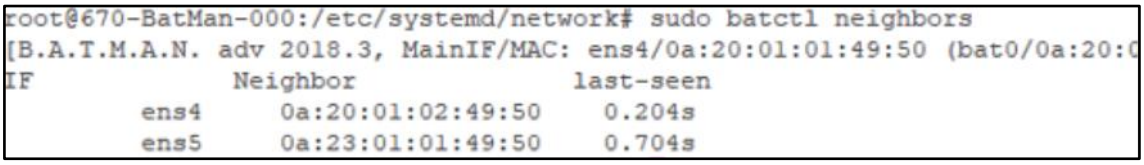
5.4. ábra 670-BatMan-000 gép B.A.T.M.A.N. interfészei

Hajtsuk végre ezt a műveletet az összes többi gépen is. Hogyha sikeresen jártunk el akkor az eszközöknek már látniuk kell egymást és egymást közt küldeniük kell az OGM üzeneteket. Ezt könnyedén ellenőrizhetjük a ***batctl o*** paranccsal.



5.5. ábra 670-BatMan-000 eszköz OGM üzenetei

Ilyenkor a protokoll automatikusan kiválasztja egyik bat hálózatban szereplő interfészét és annak szemszögéből írja le az oda érkező OGM üzeneteket. Jelen esetben az *ens4* interfész szemszögéből látjuk, ezért szerepel a képen 7 interfész. Mint ahogy azt említettem a B.A.T.M.A.N. MAC címeket használ így itt is ezt láthatjuk. Láthatunk még egy last-seen oszlopot, ahol az az eltelt idő szerepel, ami óta az utolsó üzenet érkezett. Amennyiben egy bizonyos idő óta nem érkezik üzenet valahonnan a rendszer automatikusan eltávolítja a táblájából. Ezt az időtartalmat nem tudjuk állítani alapértelmezett érték. További elemzéshez megtekinthetjük még az egyes bat csomópontok szomszédsági tábláját, ahol látjuk, hogy egyes interfészeknek kik a közvetlen szomszédjaik. [8]



5.6. ábra 670-BatMan-000 eszköz szomszédsági táblája

Ismét az *ens4* interfész szemszögéből látjuk a hálózatot így szomszédként megjelenik ugyanazon a gépen található *ens5* interfész illetve a linux bridge másik oldalán található a 3-as gépet amin az *ens5* interfész 0a:23:01:01:01:49 MAC címmel rendelkezik.

Próbáljuk meg elérni a 0-ás gépről az 1-es gép bat0 interfészét. A ping sikeres volt tehát a kommunikáció jól működik a két gép között. Járjunk el hasonló módon a többi gép közötti kapcsolat tesztelése érdekében. Amennyiben minden ping sikeres volt térjünk át az útvonalak, routing tesztelésére. Ha esetleg hiba volt a hálózatban a fenti debug parancsok segítségével kideríthetjük melyik interfész van rosszul beállítva.

5.3. Batman-adv protokoll tesztelése

Az útvonalak tesztelésére a batman-adv-ba alapértelmezetten traceroute parancsot fogjuk használni, ami kiírja, hogy milyen csomópontokon ment keresztül a csomag cél állomás felé. Próbaképpen 0-ás és a 2-es gépek közötti kapcsolatot teszteljük mivel ezek nem közvetlen csomópontok. Így amikor az egyik briget kikapcsoljuk vélhetően a protokoll reagálni fog és másik útvonalat fogja választani.

Adjuk ki a következő parancsot ***batctl traceroute 192.168.123.4*** a 0-ás gépen.

```
root@670-BatMan-000:/etc/systemd/network# sudo batctl traceroute 192.168.123.4
traceroute to 192.168.123.4 (0a:21:01:02:49:50), 50 hops max, 20 byte packets
 1: 0a:22:01:02:49:50  0.274 ms  0.206 ms  0.231 ms
 2: 0a:21:01:02:49:50  0.250 ms  0.304 ms  0.173 ms
```

5.7. ábra traceroute teszt eredménye 0-2 gépre első állapot

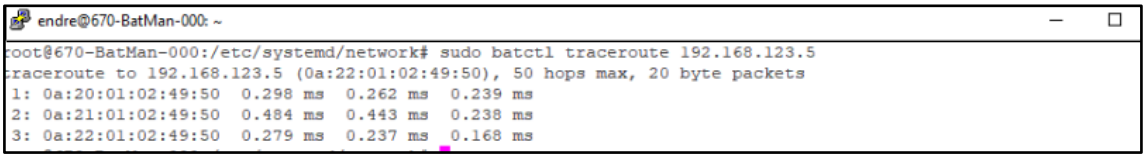
Ahogy a képen látjuk a vártnak megfelelően történt az első traceroute. 2 interfészen keresztül utazott a csomag. Az első állomás a 3-as gép *ens4* interfésze volt, majd onnan tovább ment a 2-es gép *ens4* portjára. Ezzel a csomag elérte a célállomást, így levontuk az a következtetést, hogy a protokoll a brBatMan002 es brBatMan001 bridgeket használja. Ahhoz, hogy a protokollt változtatásra kényszerítsük lefogjuk kapcsolni a brBatMan002 bridget. Ehhez a KVM hoston fogjuk kiadni a ***IP link set dev brBatMan002 down*** parancsot. Így a protokollnak másik útvonalat kell keresnie. Adjuk ki újra a ***batctl traceroute 192.168.123.4*** utasítást a 0-ás gépen és figyeljük meg merre megy most a csomag.

```
endre@670-BatMan-000: ~
root@670-BatMan-000:/etc/systemd/network# sudo batctl traceroute 192.168.123.4
traceroute to 192.168.123.4 (0a:21:01:02:49:50), 50 hops max, 20 byte packets
 1: 0a:20:01:02:49:50  0.301 ms  0.262 ms  0.236 ms
 2: 0a:21:01:02:49:50  0.299 ms  0.202 ms  0.131 ms
```

5.8. ábra traceroute teszt eredménye 0-2 gépre második állapot

Láthatjuk, hogy a protokoll egyből új útvonal után nézett. Most a brBatMan000 felé indult a csomag, mivel a másik irányba le volt kapcsolva a bridge interfész. Végül megérkezett hasonló módon a célállomásra. A vártnak megfelelően történt meg az útvonalválasztás, jól látható, hogy a rendszer képes reagálni egy olyan helyzetre mikor egy esetleges kapcsolat megsérül.

Miután láttuk, hogy csatorna sérülés esetén képes másik alternatívát találni egy semleges állomásra, teszteljük le hogy képes-e elérni azt a csomópontot amivel úgymond megszakadt a kapcsolata. A 3-as gép eddig a rendszer szempontjából szomszédos csomópontnak minősült viszont most nincs közvetlen kapcsolat. Tehát adjuk ki a ***batctl traceroute 192.168.123.5*** parancsot, ami a 3-as gép bat0 interfészének címe.



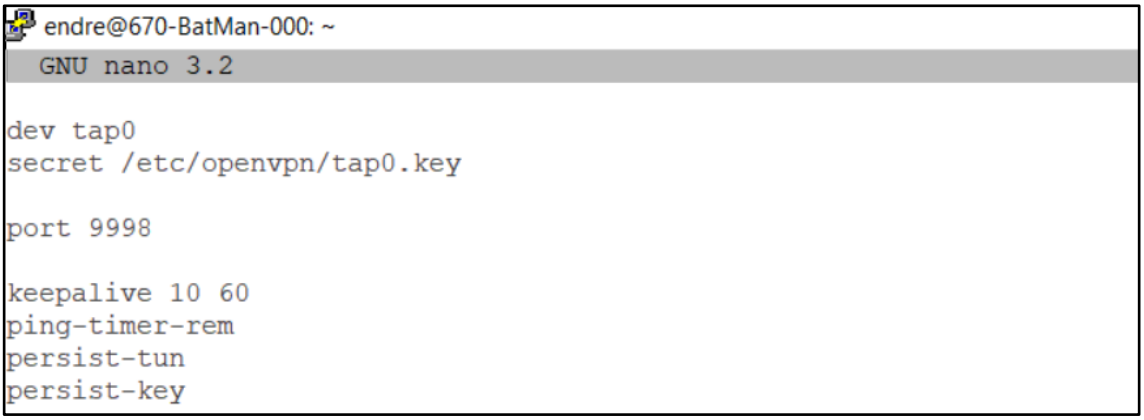
5.9. ábra traceroute teszt eredménye 0 -3 gépre

Látjuk, hogy a csomag eléri a célállomást de körbe utazik a hálózaton a közvetlen bridge hiánya miatt. Ami mutatja nekünk, hogy esetleges csatorna vesztés esetén nem esik ki az adott csomópont, ha redundánsan hozzuk létre a kapcsolatokat.

5.4. OpenVPN konfigurálása

Miután a hálózatunkat leteszteltük a batman-adv működését linux bridgekkel tovább lépünk az OpenVPN konfigurálására. Ahhoz, hogy betöltsük a modult futtassuk le az ***apt-get install openvpn*** parancsot. Először csak egy csatornát próbálunk létrehozni és azon keresztül kommunikációt létrehozni két gép között. A csatorna két végpontján egy kliens és egy szerver gép áll majd. Mindkét gép típus esetében szükségünk lesz tap virtuális interfészekre. A két tap interfész konfigurációja nem sokban különböznek egymástól. [9]

A konfigurációs fájlok létrehozásához lépjünk be a */etc/openvpn/* mappába és hozzunk létre egy *tap0.conf* fájlt.



5.10. ábra tap0.conf fájl tartalma

A képen látható, hogy megadjuk az interfész nevét, lényeges, hogy nem lehet névütközés a többi interfésszel. Továbbá megadjuk, hogy a szerver melyik portján figyeljen. A kliens majd ugyanezen porton próbál meg kapcsolatot létesíteni a másik állomással. A kliens gép konfigurációja kiegészül még egy fontos *Remote* mezővel, amiben megadjuk annak

az állomásnak az IP címét, ahova majd csatlakoznia kell a csatornával. Mindemellett a konfigurációs fájlnak tartalmaznia kell egy kulcsot és annak elérését.

Az egyszerűség kedvéért 2048 bites szimmetrikus kulcsokat alkalmazunk, amiket a következő paranccsal generálhatunk le***: openvpn --genkey --secret tap0.key***

Mindkét gépen elhelyezzük a .key kiterjesztésű fájlokat a megfelelő könyvtárba. Ezután a system-d újraindításával létrejönnek az interfészek. A létrejöttük után fel kell kapcsolnunk, illetve hozzá kell adnunk a B.A.T.M.A.N. hálózathoz. Az alábbi két paranccsal tehetjük meg ezeket: ***batctl if add tap0*** illetve ***IP link set up dev tap0***. Amennyiben sikerrel jártunk ***batctl if*** utasítást kiadva látnunk kell az aktív tap0 interfészt.

A tesztelés legegyszerűbb módja, hogyha a virtuális gépek közötti kapcsolatot pingeléssel tesszük próbára. A ping sikeres volt tehát a csatorna megfelelően működik. Így rendszerünkben a 4 gép között 3 linux bridge illetve 1 VPN csatorna működik zavartalanul.

Továbbá a mivel a rendszerünknek hibatűréssel kell rendelkeznie ezért a VPN csatorna mellé kiépítünk meg egy csatornát. Amivel ellenőrizzük, hogy a rendszer képes-e kezelni a redundáns útvonalválasztást. A második tap interfészt az elsőhöz hasonlóan hoztam létre ügyelve annak elnevezésére. Hasonlóan hozzáadtam a B.A.T.M.A.N. hálózathoz és felkapcsoltam az interfészeket. Majd megpingeltem az egyik eszközről a csatorna másik végén lévő eszközt. A ping probléma mentesen megtörtént igy a tesztet sikeresnek tekintem.

Ezzel a tesz környezetem teljes lett további tesztelésre nincs szükségem annak érdekében, hogy elkezdjem megtervezni az éles rendszert.

6. Architektúra megtervezés

Végső rendszerünk tervezésél figyelembe vettem a teszt környezetünkből leszűrt konzekvenciákat. Fontos, hogy egy rugalmas rendszert készítsünk, ami reagálni tud a csomópont hibákra az otthoni internet megbízhatatlansága miatt. Továbbá lényeges, hogy szervereink bármikor elérhetőek legyenek, rendelkezésre álljanak. Így redundánsra kell terveznünk a rendszerünket.

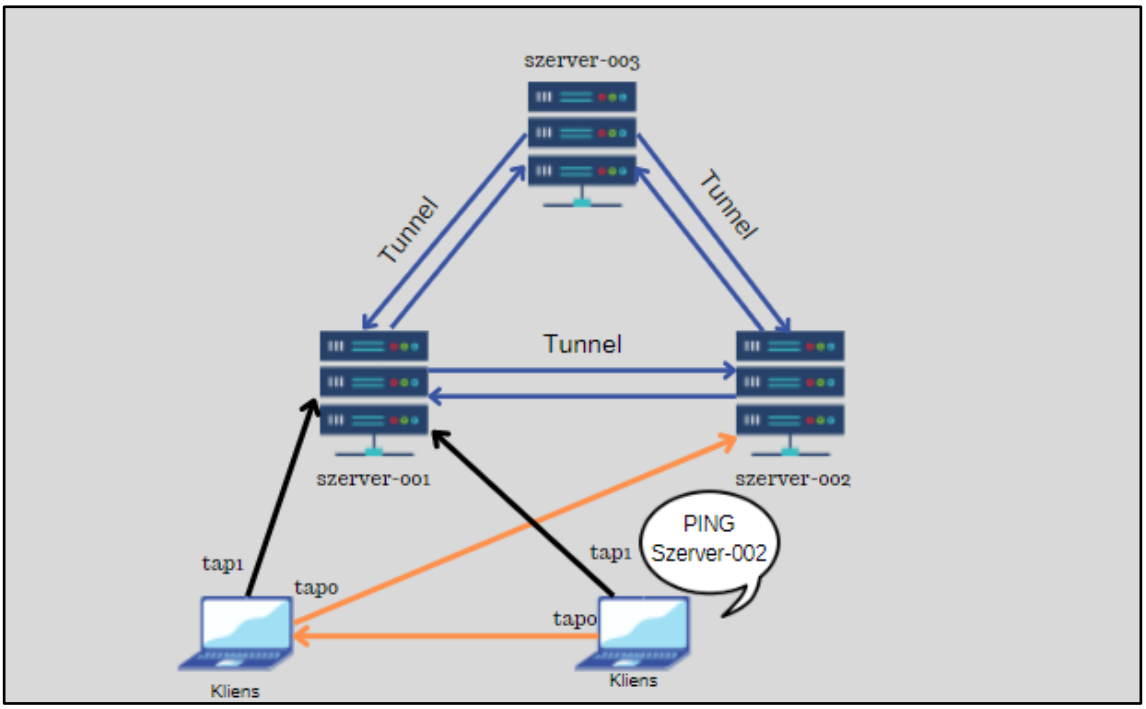
Adott 3 kiszolgáló szerver gép, amik kapcsolódnak az egyes telephelyek belső hálózatához. Mindhárom telephely egymástól földrajzilag távol helyezkedik el így nagy segítségünkre lesz az OpenVPN SSL technológiája, ami képes az interneten keresztül csatornát létesíteni a csomópontok között.

A kiszolgáló gépeknek kommunikálnia kell egymással emellett közöttük lévő kapcsolat létfontosságú, mivel ha az egyik eszköz kiesne csatorna probléma miatt, a rendszer működés képtelen lenne. Ezért a három kiszolgáló rendszer között több csatornát is üzemeltetni fogunk. Jelen esetben az áttekinthetőség miatt 2 csatorna lesz a két kiemelt forgalommal rendelkező szerver gép között.

Továbbá mivel az interneten keresztül szeretnénk elérni a rendszerünket így szükségünk lesz mindegyik géphez statikus publikus IP címekre. Amikhez majd az egyes csomópontokkal SSH kapcsolaton tudunk kommunikálni. Mikor egy végfelhasználó csatlakozik a hálózathoz kapnia kell egy belső IP-t, amivel majd a bat0 interfésze fog rendelkezni. A kapott IP A osztálybeli 10.0.0.0 címtartományból lesz kiosztva, mivel nem tudjuk előre, hogy hány eszköznek kell majd IP címet kiosztunk így a legkézenfekvőbb választás ez az osztály volt. Lehetséges lesz, hogy egy kliens gép több szerverhez is csatlakozzon egy másik csatorna segítségével. Ezzel is csökkentse a terhelést a szervergépek csatornái között és gyorsítsa a rendszert.

Amennyiben egy új eszköz csatlakozik a hálózathoz, előzetes konfigurációra van szükség mind a szerver mind a kliens gépeken. Ahhoz, hogy ez megvalósítsuk létre fogunk hozni egy menedzser gépet, ami csak ezeknek a gépeknek a konfigurálását végzi. Szakdolgozatom fontos témája, hogy ez a gép miként intézi majd az új eszközök csatlakozását a hálózathoz. A menedzser gép egy script segítségével fogja véghez vinni a telepitést. A script tartalmazni fogja az egyes konfigurációs fájlok elhelyezését és azok tartalmának helyességét. Az egyes szerverekről nyilvántartja, hogy milyen IP címeket adtak ki, illetve, hogy milyen tap interfészekkel rendelkeznek, hogy ne legyen névütközés miatt hiba a hálózatban. Továbbá mivel statikus kulcsokkal fogunk dolgozni a rendszerünkben így azt a script fogja generálni majd azt és elhelyezni mind a szerven mind a kliensen. Tulajdonképpen a script segítségével lesz egy átfogó képünk a hálózatról.

A mesh hálózatok tulajdonsága miatt az egyes csomópontok nem csak a kliens vagy szerver szerepet töltik be, hanem akár egyszerre mindkettőt. Ez megvalósítható az OpenVPN-nel mindösszesen az csatornákat kell helyesen beállítani. Így megoldható lesz az is, hogy az egyik szerverünket úgy is elérje a másik kliens eszközt, hogy neki nincs közvetlen összeköttetése vele. Emellett szeretnénk, ha a szervereink közötti csatornák csak akkor használná a rendszer, ha éppen nincs leterhelve így növeljük a sebességét a hálózatnak.



6.1. ábra VPN csatornák lehetséges megvalósítása

A szervergépek operációs rendszerének a mindenképpen Linux-disztribúciót a Linux kernel miatt. A kernel alapból tartalmazza a batman-adv protokoll és a jövőben pedig az OpenVPN is bele lesz építve. Ami azt jelenti, hogy minden Linux kernelen alapuló rendszer képes lesz kapcsolódni a hálózathoz, ideértve az okostelefonokat is. Így a három kiszolgáló gépnek Debian rendszert választottam annak is 10.11 verzióját a kompatibilitás miatt.

Összefoglalva röviden rendszerünket három kiszolgáló Debian gépünk lesz, amik a különböző telephelyeken helyezkednek el. A gépek között két OpenVPN csatorna lesz kiépítve, hogy növeljük a hálózat megbízhatóságát. A hálózatba csatlakozó gépek kapnak a 10.0.0.0 címtartományból egy IP címet. Ezt a címet használja majd ahhoz, hogy a B.A.T.M.A.N. hálózaton kommunikáljon. Az új gépek csatlakozását egy külső menedzser gép fogja intézni egy Python script segítségével.

7. Mesh architektúra megvalósítása

7.1. Host rendszer kiválasztása

A rendszer megvalósításának főszempontja volt, hogy minél jobban hasonlítson a tervezésben leírt ideális rendszerre, amely képes teljesíteni a tőle elvártakat. Főbb célom volt egy olyan rugalmas rendszer, amely képes reagálni a csomóponti hibákra. Így a teszt környezetben használt lokális KVM host rendszert egy interneten keresztül könnyedén menedzselhető architektúrára cseréltem le. Mivel az egyes csomópontok konfigurálását SSH kapcsolaton keresztül állítom be majd a script segítségével. Emellett fontos tulajdonsága a megvalósításnak, hogy változóan a felhasználók mennyiségétől képes legyen növelni a szerverek teljesítményét, fel-le skálázható legyen. Ezeknek a követelményeknek könnyedén megfelel egy felhőarchitektúra, így rendszerünk alapja egy felhőszolgáltató lesz. További előnyei a felhőszolgáltatóknak: [10]

- **Költséghatékonyság:** A felhőalapú szolgáltatások igénybevételének nincs induló költsége, a vállalkozásoknak nem kell drága hardver- és szoftvereszközöket, adattároló berendezéseket vásárolniuk. A szolgáltatás jól skálázható, a szerverek és egyéb berendezések karbantartása a szolgáltatást biztosító feladata
- **Skálázhatóság:** igény szerint beállítható, hogy mire van szükségünk, rugalmasan beállítható az előfizetés. Mit jelent ez a gyakorlatban? Azt, hogy az időben és a tárhelyben fel nem használt kapacitásért nem kell fizetnünk. Ezzel pedig ismét kikötöttünk az első pontnál, hiszen csökkentettük a vállalatunk költségeit.
- **Gyorsaság:** a napjainkban elérhető felhő üzemeltetés már rendkívül gyorsan, néhány kattintással telepíthetőek. Ennek köszönhetően perceken belül használni tudjuk őket.
- **Megbízhatóság.** A felhőalapú számítástechnika egyszerűbbé és olcsóbbá teszi az adatmentést, a katasztrófa utáni helyreállítást és az üzletmenet folytonosságát, mivel az adatok a felhőszolgáltató hálózatán több redundáns helyszínen is tükrözhetők.

A három Deployment modell közül a publikus felhő megoldás fogom választana mivel a nem áll hardver rendelkezésemre, illetve az üzemeltetéssel sem kell ez esetben foglalkozni. Így anyagi szempontból ez a modell a legjobb választás. A szolgáltatói modellek közül IaaS (Infrastructure as a service) azaz platform, mint szolgáltatást fogom választani mivel ebben az esetben a szolgáltató maga biztosítja a teljes mögöttes platformot, operációs rendszert. Emellett fontos számunkra, hogy megoldja a hálózatot, illetve a tárolást is. Ebben az esetben nekünk csak az alkalmazásunkat kell feltelepíteni de az egyszerűség kedvéért az alkalmazásunk egy külső gépről fog futni és SSH kapcsolaton keresztül fogja beállítani, ezt a későbbiekben még részletezni fogom.

A következő lépésünk, hogy kiválasztjuk melyik szolgáltatónál szeretnénk a szolgáltatásunkat futtatni. Több nagy szolgáltató közül választhatunk, de végül nekem a Huawei felhőszolgáltatóra esett a választásom, tekintve, hogy viszonylag új szolgáltatóról

beszélünk európában remek technológia újításokkal, illetve a nagyobb versenytársakhoz képest kedvezőbb árakkal.

7.2. Erőforrások kiválasztása

Következő részben ki kell választanunk a számunkra megfelelő erőforrások, amik képesek ellátni a nekik szánt feladatot mindezt a lehető legjobban optimalizálva anyagi és terhelési szempontból.

Első sorban szükségünk lesz 5 darab virtuális gépre, amik a topológiánk alapját fogják adni. Ezeket a felhószolgáltatónál ECS-nek az az Elastic Cloud Server-nek nevezik. Az ECS egy alapvető számítási egység, amely vCPU-kból, memóriából, operációs rendszerből és Elastic Volume Service (EVS) kötetekből áll. Az ECS létrehozása után hasonlóan egy fizikai rendszerhez egyből használatba is vehetjük a gépeket. Könnyedén újraindíthatjuk, fel-le kapcsoljuk, vagy akár az erőforrásokat is módosíthatjuk. [11]

Miután kiválasztottuk a nekünk megfelelő erőforrásokat meg kell adni a VPC (Virtual Private Cloud) beállításokat. Ez a szolgáltatás képes a gépeket logikailag elszigetelni emellett virtuális privát hálózatokat hozhatunk benne létre. Továbbá testre szabhatjuk az alhálózatokat, a biztonsági csoportokat, hálózati ACL-eket. Mi rendszerükhöz a 192.168.0.0/16 C osztályú IP tartományt választottam. Ügyelnünk kell arra, hogy amikor majd telepítjük a rendszerünket ott bat0 hálózati interfésznek kell adnunk egy IP címet aminek nem szabad ütköznie ezzel a tartománnyal.

Name/ID	Monitori...	AZ	Status	Specifications/Image	IP Address
eg-OPEN-VPN-Server003 9741cb70-f9e0-46c9-8c...		AZ1	Running	2 vCPUs | 4 GiB | c6s... eu-ubuntu-20.04-x86_	[redacted] (EIP) 5 Mbit/s 192.168.1.147 (Private IP)
eg-OPEN-VPN-kliens001 4e3d0707-8d93-467e-8...		AZ2	Running	2 vCPUs | 4 GiB | c6s... eu-ubuntu-20.04-x86_	[redacted] (EIP) 10 Mbit/s 192.168.1.194 (Private IP)
eg-OPEN-VPN-kliens002 803e218c-891e-4992-a0...		AZ1	Running	1 vCPUs | 4 GiB | s6.... eu-ubuntu-20.04-x86_	[redacted] (EIP) 5 Mbit/s 192.168.1.219 (Private IP)
eg-OPEN-VPN-server001 56b188f4-cd54-41bf-ad...		AZ1	Running	4 vCPUs | 8 GiB | c6s... eu-ubuntu-20.04-x86_	[redacted] (EIP) 5 Mbit/s 192.168.1.124 (Private IP)
eg-OPEN-VPN-server002 be4bb8a7-3c1c-4072-b4...		AZ1	Running	8 vCPUs | 32 GiB | s6... eu-ubuntu-20.04-x86_	[redacted] (EIP) 1 Mbit/s 192.168.1.223 (Private IP)

7.1. ábra Erőforrások általános specifikációja

Az alapvető beállítások után hozzuk létre az ECS-ket. A rendszer kialakítását tekintve 3 szerver gépünk lesz, illetve 2 darab kliens csomópontunk. Mindegyik virtuális géphez beállítottunk EIP (Elastic IP) – azaz publikusan elérhető IP címet – aminek segítségével elérhető válik az interneten keresztül. Fontos szempont ez számunkra hiszen a gépeket SSH kapcsolaton keresztül szeretnénk konfigurálni, amit a későbbiekben még részletezni fogok. Az ECS-ek létrehozása után első lépésünk lesz, hogy az SSH kapcsolatot teszteljük majd beállítsuk a kulcsot. Ehhez nyissunk egy parancssort majd lépjünk be a gépekere az

EIP-ken keresztül. Az első bejelentkezésnél jelszót fog kérni a rendszer, amit le kell cserélnünk RSA kulcsokra mivel a program ennek a segítségével létesít kapcsolatot. Az aszimmetrikus kulcsú titkosítás nagy előnye, hogy az interneten keresztül is történhet, mivel attól mert valaki megszerzi a nyilvános kulcsunkat még nem képes megfejteni a tikosított üzentet a privát kulcs nélkül. Ennek a beállítása után a gépek készen állnak a telepítésre.

7.3. IMOGO Script leírása

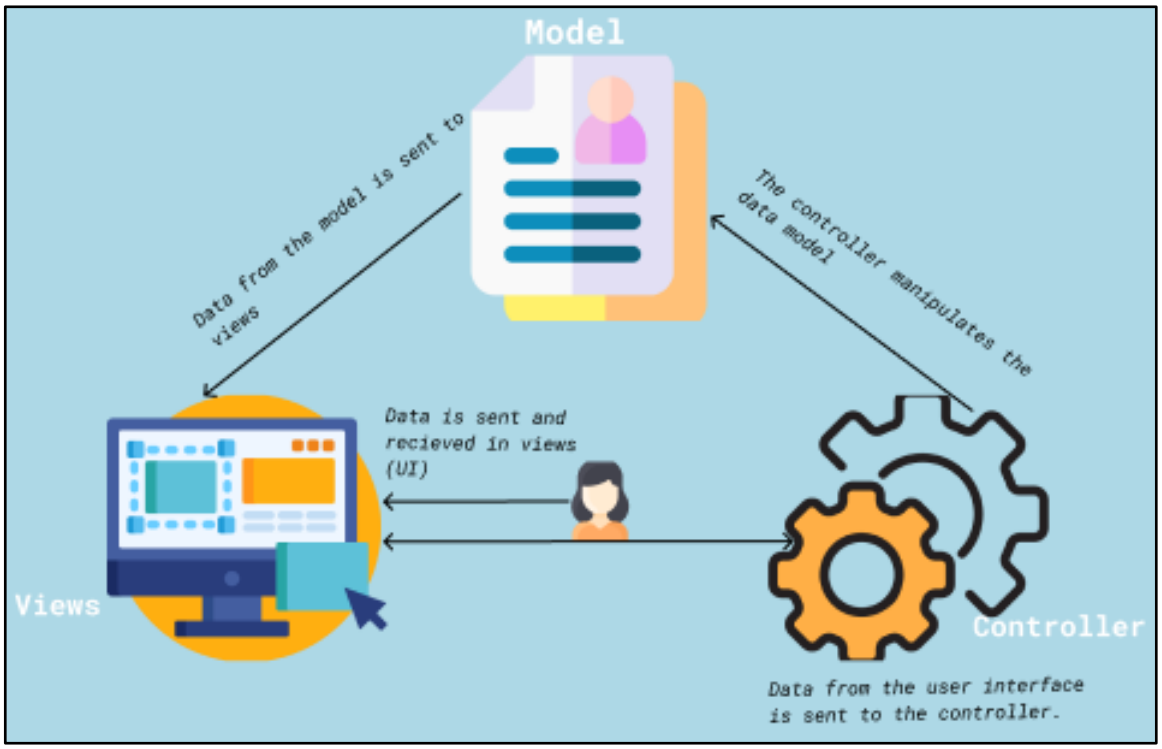
Mivel a projekt vállalati környezetben készült és több külső komponenst felhasználunk így célszerű volt a saját programom elnevezni így kapta az IMOGO nevet.

Miután elkészültünk az erőforrások beállításaival áttérhetünk a dolgozatunk fő céljához, hogy létrehozzuk a gépek közt azt a kapcsolatot, ami képes biztonságos kommunikációt létrehozni a gépek között az interneten keresztül. Mivel szeretnénk egyszerre több gépet beállítani, emellett új kapcsolatot hozzáadni a hálózathoz, ezen felül fel-le kapcsolni az egyes kapcsolatokat rengeteg időt spórolhatunk meg ha ezeket a folyamatokat automatizáljuk. Az egyes gépek konfigurációja egészen hasonló csak apró változtatásokat kell belevinni így célszerű egy scriptet írni ehhez. Erre a célra a legalkalmasabb nyelv a Python mivel egy script nyelv, amely emellett rengeteg modullal van ellátva.

Az alapvető elvárásunk a script felé, hogy egy egyszerű felhasználó felületen keresztül vezérelni lehessen az előre összeállított bash scripteket. Az egyes scriptek segítségével képes legyen felkonfigurálni egyszerre több csomópontot is. Alkalmas legyen új csomópont hozzáadására, illetve meglévő kapcsolat lekapcsolására. Emellett megjelenítse az aktuális topológiát gráfok segítségével.

7.3.1. Program általános felépítése

Mivel a python egy script nyelv így lényeges, hogy a kódot jól áttekinthetően erősen strukturáltan írjunk meg. Lényeges, hogy az egyes programrészek kellőképpen elkülönítsük egymástól ezzel segítve az olvashatóságot. Ehhez MVC (Model-View-Controller) tervezesési mintát fogom alkalmazni.



7.2. ábra MVC tervezési minta [18]

Ez az architektúra az alkalmazást három részre osztja, amelyek egymástól függenek és kapcsolódnak egymáshoz. Egyes részeinek jellemzői a következők:

- **Modell:** A modell komponens az alkalmazásunk alapját képezi itt határozzuk meg a használni kívánt adat típusokat, itt tároljuk adatainkat. Tekinthetünk rá úgy, mint az architektúra központi részére, parancsokat kap a Controller rétegtől majd adatokat szolgáltat a View réteg felé. Viszont a felhasználói felülettől nem fogad el utasításokat kizárólag a Controller-rel kommunikál.
- **View:** Ez a komponens felel a felhasználói felületen megjelenítendő adatokért, illetve a felhasználóval való összes interakcióval. Ideértve a bementeket az egyes ablakok kinézetét, megjeleníti a Controller, illetve a modell által kapott adatokat. A beérkező adatokat pedig továbbítja a Controller felé.
- **Controller:** A Controller komponens a felelős a Modell, illetve a View összeköttetéséért. Ez feleltethető meg az alkalmazásunk agyának itt található az üzleti logika. Ez felel a View-ból érkező kéréseknek a paranccsá alakításáért majd azokat végre hajtja a Modell elemein. [12]

7.3.2. Modell réteg kialakítása

Első sorban programunk alapját képező modell réteget szeretném kialakítani úgy, hogy jól áttekinthető kellően elszeparált osztályokat hozok létre. A hálózatunkat felmérve előreláthatóan 3 nagyobb egységre fogom bontani a rendszert. A kapcsolatok kialakításához az először is OpenVPN nek Tap interfészekre van szüksége, ennek beállításához létre hoztam a *tap* osztályt, ami az alábbi módon épül fel:

```
class tap:
    def __init__(self,type,port,sLink,dLink,key="",remoteIP = ""):
        self.type=type
        self.port=port
        self.sLink=sLink
        self.dLink= dLink
        self.remoteIP= remoteIP
        self.key = key
```

7.3. ábra TAP osztály konstruktora

Ahogy a képen is látható a konstruktorán keresztül kapja meg a konfigurációhoz szükséges adatokat. Elsőként megkapja a típusát az interfész, hogy kliens vagy szerver kapcsolódási pont lesz e. Ennek értéke string típusú s vagy k lehet. A soron következő elem a tap portja, amin keresztül a kliens keresni fogja a szerver fordítva pedig ott fog hallgatni a kiszolgáló. Ennek a paraméternek fontos, hogy egyezzen az értéke a másik csomóponton lévő porttal, különben nem fog létrejönni a kapcsolat. A következő két paraméter *slink,* illetve *dlink* a megjelentést fogja segíteni, ez határozza meg a gráf éleinek végpontjait. A következő paraméter a *remoteIP,* amit csak a kliens tap konfigurációknál kell megadni mivel csak a nekik van ilyen mezőjük. Ennek értéke a szerver csomópontnak EIP értéke lesz. Az utolsó érték pedig *Key* lesz, ami a kapcsolatnak a statikus kulcsa lesz. Itt is figyelemmel kell lenni, amikor példányosítjuk az osztályokat akkor az összetartozó tap interfészek azonos statikus kulccsal rendelkezzenek.

```
class node:
    def __init__(self,id,batmanIp,Name):
        self.id=id
        self.batmanIp=batmanIp
        self.Name=Name
        self.neighbors = []
        self.NOTap = 0
```

7.4. ábra Node osztály konstruktora

Következő osztályunk a *Node* osztály lesz, ami a hálózatunk szempontjából egy-egy csomópontot fog jelölni. Itt is előzőhez hasonlóan egy egyszerű osztályról beszélünk, amely konstruktorán keresztül kapja meg az alábbi értékeket:

Első paramétere az *id* mező, amely a csomópontokat egy egyedi azonosítóval lát el, ami alapján később szűrhetünk rájuk. Következő mező a *batmanIP* lesz, ami az egyes gépeknek a Batman hálózaton használt bat0 interfészén használt IP címe lesz. Itt is fontos megjegyezni, hogy ügyelnünk kell rá, hogy ez az IP ne ütközzön az felhő VPC-ben használt IPtartománnyal. Soron következő elemünk a *Name* paraméter lesz, ami a megjelenítésben játszik majd nagy szerepet hiszen a gráf egyes csúcsai ezen a néven fognak megjelenni. Ezt követi az adattagok sorában a *neighbors* mező, ami egy üres listaként hozunk létre a példányosítás során. Ez a lista az egyes csomópontoknak fogja a szomszédos gépeit tartalmazni. A korábbiakban meghatározott *Tap* osztály segítségével. Mivel egy Tap interfész két gép közötti kapcsolatot hivatott jelképezni így tökéletesen alkalmas erre. Így minden *Node* képes lesz számontartani az aktuális kapcsolatait mivel az egyes Tap elemekből tudni fogja a cél, illetve a forrás csomópontok egyedi azonosítóit. Az utolsó mező pedig *NOtap* egy szimpla adattag, ami a megjelenítés során lehet hasznos számunkra, hogy a gráfunk egyes éleinek súlyát jelölje.

Az utolsó osztályunk pedig a *Topologie* osztály lesz, ami különböző a másik két osztálytól mivel a példányosítás során egyetlen adattagot sem kell megadnunk:

```
class Topologie:
    def __init__(self):
        self.Nodes=[]
```

7.5. ábra Topologie osztály konstruktora

Mindösszesen egyetlen üres listát hozunk létre a példányosítás során, ami a program futása közben létrehozott összes Node típusú elemet tárolni fogja. Így ennek az osztálynak segítségével az egész hálózatról képet kaphatunk. Ez a vizuális megjelenésben nagy segítségünkre lesz, hiszen a listának az elemeit kell majd ábrázolnunk. Ahhoz, hogy későbbiekben a gráfot összerakjuk szükségünk lesz belső függvényekre, amiket logika szerint két csoportra osztanék. Az első csoport, ami gráf felépítését szolgálják, a második pedig ami ezeknek segítségével megjeleníti a gráfot.

```
    def Append(self, new_value):
        self.Nodes.append(new_value)

    def addNode(self,net,id,label):
        net.add_node(id,label)

    def addEdge(self,net,source,destination,val=2):
        net.add_edge(source,destination,value=val)

    def createTopologie(self):
        net = Network()
        return net
```

7.6. ábra Topologie osztály függvényei 1

Az első metódusunk egy egyszerű *Append()* függvény, ami arra szolgál, hogy a létre hozott Topologie osztály Nodes listáját fogja feltölteni Node elemekkel. Ehhez a függvény bementeként megkapja a hozzáadni kívánt Node példányt. A következő három függvény a gráf egyes részeit fogja összerakni. A megjelenítéshez használni fogjuk a *pyvis.network* bővítményt, ami képes hálózati gráfok építésére és vizualizálására. Csomópontonként vagy élenként testre szabható. A csomópontoknak színeket, méreteket, címkéket és egyéb metaadatokat lehet adni. [13] Ennek a csomagnak a beépített függvényeit fogjuk használni, majd a későbbiekben a Controller osztályban meghívni. Az *addNode()* illetve *addEdge()* egészen hasonló metódusok mivel mindkettővel bővítjük a gráfot. Az egyik egy élet a másik pedig egy csomópontot ad hozzá. Paraméterként fogják megkapni az egyes metaadatokat, amiket egy Network objektumhoz fogja hozzáadni. A *CreateTopologie()* függvényt ezt az előre definiált Network osztály fogja létre hozni, amihez a későbbiekben az éleket, illetve a csomópontokat fogjuk hozzáadni.

A második csoportunk a gráf megjelenítését fogja szolgálni. Ehhez szüksége lesz az előbb definiált Network osztályra, amit már feltöltöttünk csomópontokkal majd azokhoz beállítottuk a kapcsolatokat. Ezt az értéket bementi paraméterként fogja megkapni az alábbi módon:

```
def generateGraph(self,network):
    network.show('nodes.html')
    hti = Html2Image()
    hti.screenshot( html_file='nodes.html', save_as='nodes.png')
```

7.7. ábra Topologie osztály függvényei 2

A kapott értkéken meghívjuk az előre definiált *show()* ami a gráf megjelenítése szolgál. Alapértelmezetten egy html fájlba generálja ki a rendszer, ami nekünk a későbbiekben problémát okozhat mivel szeretnénk felhasználói felületen megjeleníteni azt. Ezért a html

oldalból a html2Image bővítmény segítségével egy png fájt generálunk, majd lementjük. Ezt fogjuk a későbbiekben jeleníteni a képernyőn.

7.3.3. View réteg kialakítása

A következő réteg, ami bemutatásra kerül a View felépítése lesz. Mivel programom fő célja, hogy a hálózatot konfigurálja így a felhasználói felület csak azt a célt szolgálja, hogy minél egyszerűbb legyen a scriptek használata. Ennek értelmében törekedtem az letisztultságra, illetve felhasználóbarát kialakításra. Az alapgondolat, hogy a program indításakor a felhasználó egyből a főmenüben találja magát, ahol egyből láthatja az alap funkciókat egymás alatt. Majd innen nyílik tovább majd az egyes ablakok a rendeltetésüktől függően. Ennek a felépítésnek mentén szeretném bemutatni a réteget.

Ennek a felépítésnek összeállítására létre hoztam egy alap *screen* osztályt, amely tartalmazni fogja minden ablakunkat. Ennek az osztálynak a különböző függvényei fogja létrehozni az egyes mellékablakokat. A megjelenítés a *tkinter* nevezetű python bővítménnyel valósítom meg. A modernebb megjelenítés érdekében egy tovább fejlesztett verzióját fogom használni a *customtkinter- t*.

Az osztály létrehozása után el is kezdhetünk definiálni a konstruktort, ami egy Main ablak objektumot fog majd kapni paraméterként, amin majd a módosításokat végre fogja hajtani. Első sorban a bővítmény beépített függvényeit fogjuk használni a megjelenítés során majd azok segítségével hívjuk meg a Controller rétegben lévő metódusokat. Az ablak fő paramétereit az osztály elején meghatározzuk. Beállítjuk a geometry() függvény segítségével az ablak nagyságát pixelben értelmezve. Ami jelen esetünkben 260*320 lesz, mivel a kis méret lehetőséget ad majd a későbbiekben hogy telefonokra is kiterjesszem az alkalmazást. Ez után elnevezzük az ablakunkat „Főoldal” névvel, majd beállítjuk az alapvető stílusát a customtkinter segítségével dark illetve green színkombinációra. Ezzel is modern külsőt kölcsönözve alkalmazásunknak. Mivel ez lesz a kezdő oldala alkalmazásunknak így be kell állítanunk neki a *toplevel* értékékének a nullát. A későbbiekben lesz ez segítségünkre, hogy újabb ablakokat indíthassunk ezzel párhuzamosan.

```
class screen:
    def __init__(self, Main):
        self.Main = Main
        self.Main.geometry("260x320")
        self.Main.title("Főmenü")
        customtkinter.set_appearance_mode("dark")
        customtkinter.set_default_color_theme("green")
        self.toplevels = 0
        button1 = customtkinter.CTkButton(Main, text="Topológia megjelenítése", width=200,
                                  height=40,text_font="Timesnewroman", command=self.display_topologie)
        button1.place(x=30, y=100)
        button2 = customtkinter.CTkButton(Main, text="Hálózat konfigurálása", width=200,
                                  height=40,text_font="Timesnewroman", command=self.setUpT)
        button2.place(x=30, y=30)
        button3 = customtkinter.CTkButton(Main, text="Új csomópont hozzáadása", width=200,
                                  height=40,text_font="Timesnewroman", command=self.New_Node)
        button3.place(x=30, y=170)
        button4 = customtkinter.CTkButton(Main, text="Csomópont leválasztása", width=200,
                                        height=40, text_font="Timesnewroman", command=self.Del_Node)
        button4.place(x=30, y=240)
```

7.8. ábra screen osztály megvalósítása

Következő lépésünk az ablakon szereplő widgetek felhelyezése lesz a képernyőre. Mivel a kezdőképernyő csupán a navigációt szolgálja a többi ablak között így roppant egyszerű a felépítése. Először egy gombot hozunk létre úgy, hogy létrehozunk egy változót, amiben eltároljuk majd a gomb egyes tulajdonságait. A customtkinter egy beépített CTKButton metódusát használva létrehozzuk a widgetet. Itt megadjuk neki először, hogy melyik ablakon szeretnénk megjeleníteni, ami mi esetünkben a Main lesz. Emellett megadjuk a gomb feliratát, szélességét, magasságát, betűtípusát, illetve a gomb aktiválásál használni kívánt metódusokat.

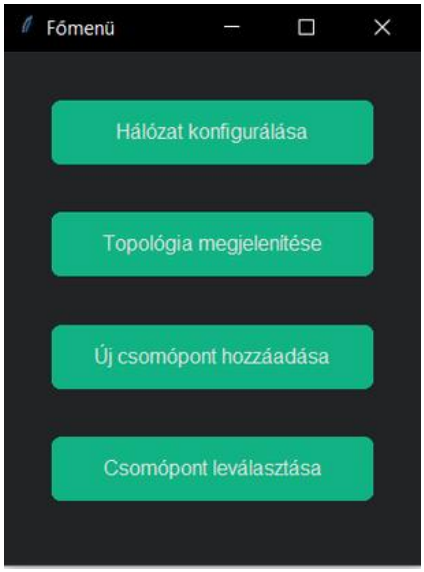
Ennek mintájára létrehozzuk a jövőben használni kívánt metódusokhoz gombokat. Előre láthatólag 4 gombra lesz szükségünk. Ezek a függvények fogják megnyitni az újabb ablakokat, ahol különféle események fognak megvalósulni. Úgy, mint topológia megjelenítése, teljes hálózat konfigurálása, új csomópont hozzáadása, illetve a csomópont leválasztása. A megfelelő függvények neveit a *commad* mezőértékének kell megadni. A gombok létrehozása után csupán egy dolgunk maradt, hogy megjelenítsük a magát az ablakot a mainloop() beépített funkcióval.

```
if __name__ == "__main__":
    Main = customtkinter.CTk()
    obj = Mainscreen.screen(Main)
    Main.mainloop()
```

7.9. ábra Main window létrehozása

Az egész program igazán egyszerűen ezzel a három sor futtatásával indul el. Amit a struktúra megőrzése érdekében az egyes rétegektől függetlenül egy __init__ fájlban

kerül meghívásra. Itt létrehozunk egy customtkinter ablakot majd azt odaadjuk egy screen típusú osztály konstruktorának. Majd a programot futtatva ezt az ablakot kapjuk:



7.10. ábra Főmenü ablak megjelenítése

Ha kipróbáljuk az egyes gombokat akkor egyelőre a program hibát fog dobni mivel még nem létező függvényt próbál meghívni. Ezt a későbbiekben orvosoljuk. Tovább haladva a View réteg bemutatása során mivel az ablakok kinézete igen hasonló nem fogom egyesével részletezni azok dizájn elemeit, csak az adott ablakra jellemző tulajdonságot fogom kiemelni.

A soron következő ablak feladata a topológia megjelenítése lesz.. Viszonlag egyszerű de annél szemléletesebb ablakról beszélünk mivel csak egy képet fog megjeleníteni a képernyőn. A leirása a következő képpen néz ki:

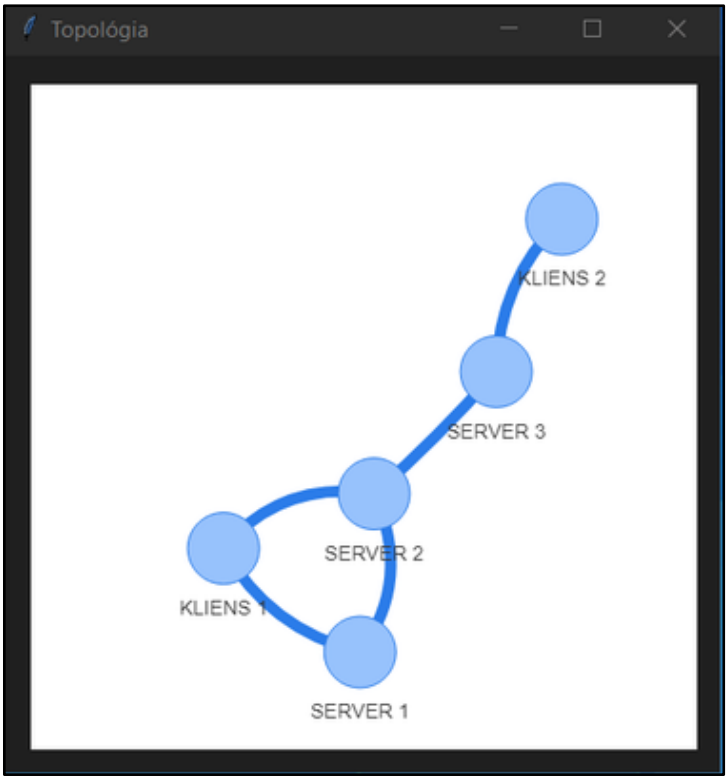
```
def display_topologie(self):
    newwin = customtkinter.CTkToplevel(self.Main)
    newwin.title('Topológia')
    newwin.geometry("500x500")
    canvas = customtkinter.CTkCanvas(newwin, width=460, height=460)
    canvas.place(x=20, y=20)
    canvas.configure(bg='black')
    CreateTopologie.Generate()
    self.img = PhotoImage(file="nodes.png")
    canvas.create_image(-7, -7, anchor=NW, image=self.img)
```

7.11. ábra Topológia ablak leírása

Elsőként egy új ablakot kell létrehoznunk, amit az eddigiektől eltérően a CTkToplevel() függvénnyel teszünk meg. Ennek segítségével úgymond az előző main ablak fölé tudunk egy új független window objektumot létrehozni. Majd hasonlóan az main ablakhoz

beállítjuk az alapvető paramétereket. Ez után létre kell hoznunk egy Canvas elemet, ami a kép megjelenítésére használunk. Itt megadjuk, hogy melyik ablakon hozza létre a widgetet, illetve annak szélességét és magasságát., majd elhelyezzük a vásznunkat a beépített place() függvény segítségével a megadott pozícióba.

Ahhoz, hogy megkapjuk az előre le generált kész topológiánkat le kell futtatnunk a Controller osztályból a megfelelő függvényünket. Miután legeneráltuk a topológiánkat létre kell hoznunk egy *PhotoImage* osztályt, amiben megadjuk a megjelenítendő fájl nevét. Majd ezt kimentve egy változóba alapvető beállítások mellett, megadjuk a Canvas widget *Image* paraméterének értékeként. Az program futtatása során az ablak a következőképpen néz ki:



7.12. ábra Toplógia ablak megjelenítése

Soron következő ablakunk funkciója a teljes hálózat konfigurálása lesz. Ez egy kicsit eltérő az eddigi ablakainkhoz képest mivel a függvény meghívásakor nem jelenik meg új ablak ameddig a hálózat csomópontjait be nem állította a rendszer. Ahogy a 7.13 képen is látható, meghívjuk a Controller rétegből ConfigureConnection.py fájl *factory()* metódusát, ami bash parancsok segítségével beállítja az egyes gépeket. A Controller réteg leírásakor majd bővebben kifejtem ezt. Ammenyiben ez a függvény True értékkel tér vissza akkor a hálózatunk sikeresen felépült, ellenkező esetben hiba lépett fel a konfiguráció során, melyről részletesebb leírás található a naplófájlokban. Mindkét esetben egy ablak fog megjelenni, ahol az eredménytől függően kapunk majd visszajelzést.Mindkét ablakon 2 gomb található, amelyeknek felirata bezárás, illetve Log megnyitása. Az elnevezések beszédesek lehetnek hiszen az első gomb az ablakot bezárja

majd visszalép a főmenübe. A másik viszont egy újabb ablakot fog meghívja az osztály LogWindow metódusát, ami képes megjeleníteni az egyes bash parancsok eredményeit. Mivel ez az ablak majdnem minden hálózaton történő esemény után megjelenik ezért szeretném bemutatni működését.

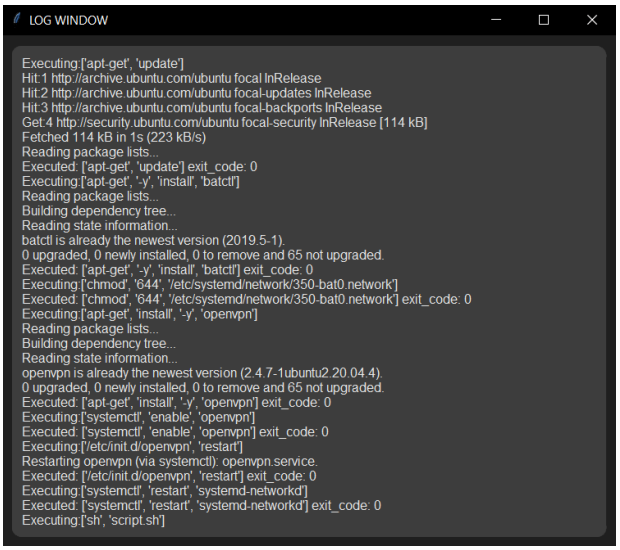
```
if ConfigureConnections.factory() == True:
    lab = customtkinter.CTkLabel(display,text_font="Timesnewroman"
                                 , text="Hálózat sikeresen konfigurálva!")
    but = customtkinter.CTkButton(display, text="Log megnyitása"
                                  ,command=self.LogWindow, bg="grey",width=110,height=30)
    but2 = customtkinter.CTkButton(display, text="Bezárás"
                                   , command=display.destroy, bg="grey",width=110,height=30)
    lab.place(x=50, y=20)
    but.place(y=75, x=30)
    but2.place(y=75, x=150)
else:
    lab = customtkinter.CTkLabel(display, text_font="Timesnewroman",
                                 text="HIba lépett fel a hálózat konfigurálása során!")
    but = customtkinter.CTkButton(display, text="Log megnyitása"
                                  , command=self.LogWindow, bg="grey", width=110,
                                  height=30)
    but2 = customtkinter.CTkButton(display, text="Bezárás", command=display.destroy, bg="grey", width=110,
                                   height=30)
    lab.place(x=50, y=20)
    but.place(y=75, x=30)
    but2.place(y=75, x=150)
```

7.13. ábra Teljes hálózat beállítása ablak leirása

Az alap program, aminek segítségével a Shell parancsokat futtatom SSH kapcsolaton keresztül rendelkezik egy beépített parancssori kimenettel minden parancs után. A program kezdeti fázisában ilyen formában kaptunk visszajelzést a lefutott sorokról. Annak érdekében, hogy a felhasználónak ne kelljen kilépni a programból és futás közben lehetősége legyen visszaolvasni a parancsok lefutásának eredményét a kijelzőn némi módosítást kellett eszközölni.

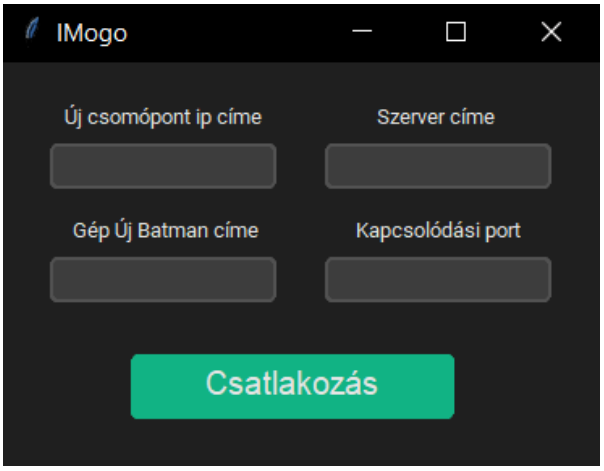
Eredetileg egy egyszerű print metódust használtam, ami egy sorba összefűzte az adott parancs tartalmát majd annak kimenetelét és dátumát kiírja a fejlesztőkörnyezet kimenetére. A program megfelelően működött addig amíg a nem használtuk a Tkinter megjelenítőt. Mivel a bővítmény a Mainloop() függvényt meghívva létrehoz egy ablakot, ezért annak bezárásáig a program nem lép tovább a kódsoron. Így tegyük fel, hogy az első sor lefuttatása után meghívjuk a log ablakot, amiben megjelenik az összerakott szöveg de a program futása megáll addig ameddig az az ablak él. Így az nem lép tovább a következő végrehajtandó sorra. Ennek megoldására két megoldást találtam. Elsőként, hogy minden egyes megnyitott log ablak után automatikusan egy bizonyos idő eltelte után az ablakot automatikusan töröljük. Ebben az esetben több problémával is szembesültem, mivel minden egyes lefutott Shell parancs után a program új ablakot hozna létre, ami rettentő erőforrás igényes feladat lenne. Emellett mivel automatikusan záródna be az ablak a felhasználói élményt erősen rontaná a megjelenítés. Így másik alternatív megoldás után néztem a megvalósításom során.

Mivel a program futása közben nem tudtam frissíteni az ablakot így olyan megoldás kellett, ami eltárolja az összes eddigi parancs eredményét. Így amikor befejeződik egy utasítás a program megnyitja az erre a célra létrehozott *log.txt* fájlt és annak kimenetelét tartalmazó szöveget beilleszti oda. Majd miután a parancssorozatok végére ért létrehozza a LogWindow ablakot, így nem akadályozva annak futását. A LogWindow ablak tartalma pedig *log.txt* sorai fognak belekerülni. A megjelenítésre most egy Textboxot fogunk használni, aminek paramétereit személyre szabhatjuk a widget létrehozásakor. Mivel ezt az ablakot univerzálisan használjuk több konfiguráló függvény esetében így amikor új folyamatot kezdünk töröljük a log fájl tartalmát.



7.14. ábra LogWindow ablak megjelenítése

A View réteg következő eleme új csomópont létrehozásáért lesz felelős. Az eddigi megvalósításoktól abban fog különbözni, hogy itt nem előre definiált adatokból dolgozunk, hanem felhasználói bementet kell kezelni. Majd annak az értékeivel kell meghívni a Controller réteg függvényeit.



7.15. ábra Új csomópont hozzáadása ablak megjelenítése

Ehhez a megszokott módon létrehozunk a screen osztály egy új metódusát, ami definiálja az ablakot. Itt is Toplevel() beépített függvényt fogjuk használni ahhoz, hogy főablak mellett egy tőle független elemet hozzunk létre. Ezek után beállítjuk az ablak alapértékeit majd elkezdjük a widgetek meghatározását. Az alapvető feliratokon kívül szükségünk lesz CTkEntry() widgetekre is, amik felhasználói inputot fogják kezelni. Mivel 4 bemeneti értékünk lesz így szükségünk van 4 előre definiált StringVar() változóra, amibe az értékeket menteni fogjuk. Amikor az entry widget definiálásra kerül akkor annak *textvariable* paraméterként megadjuk a változók neveit. Majd az ablak aljára létre hozunk egy „Csatlakozás” felirattal ellátott gombot, ami az adatok begyűjtéséért fogja végezni az alábbi módon:

```
def collect():
   ConfigureConnections.singleConnection(TC.get(), TB.get(), TP.get(),  TR.get())
   addscreen.destroy()
   okwindow = customtkinter.CTkToplevel(self.Main)
   okwindow.geometry("290x130")
   okwindow.title('OKE')
   lab = customtkinter.CTkLabel(okwindow, text_font="Timesnewroman", text="Csomópont hozzáadva a hálozathoz")
   but = customtkinter.CTkButton(okwindow, text="Log megnyitása", command=self.LogWindow, bg="grey", width=110,height=30)
   but2 = customtkinter.CTkButton(okwindow, text="Bezárás", command=okwindow.destroy, bg="grey", width=110,height=30)
   lab.place(x=20, y=20)
   but.place(y=75, x=30)
   but2.place(y=75, x=150)
register = customtkinter.CTkButton(addscreen, text="Csatlakozás",command=collect, bg="grey", width=200,
                        height=40,text_font="Timesnewroman" )
register.place(x=80, y=180)
```

7.16. ábra collect() függvény leírása

A gomb lenyomásakor a fent látható *collect()* függvény kerül meghívásra. Már az első sorban meghívjuk a Controller réteg *SingleConnection*() metódusát, ami majd a megfelelő Shell scripteket futtatja. Ennek paramétereként az entry widgetek *get()* beépített függvényét, amely a bemeneti dobozok aktuális értékével tér majd vissza string típusként. Az új csomópont beállítása után bezárjuk az aktuális ablakot majd újat készítünk helyette, mely segítségével lehetőségünk van az aktuális naplófájlok megtekintésére.

Az utolsó ablak, amiről beszélni fogok a csomópontok törlését fogja végezni. Felépítését tekintve egészen hasonló a csomópont hozzáadása ablakhoz. Mivel itt is felhasználói bemenetek alapján történik az odatartozó függvények meghívása. Jelen esetben a felhasználónak 3 adatot kell megadnia. A kliens, illetve szerver gép IP címét, mivel ennek segítségével fogja a kapcsolatokat létrehozni a Controller réteg függvénye. Emellett meg kell adnunk a tap interfész portját mivel ez alapján egyedinek mondható a kapcsolat két gép között. A hozzáadás ablakhoz hasonlóan annak alján egy gomb található, amelynek kódjának felépítése azonos. A függvény elején meghívja az *ExecuteDelete()* metódust a bemeneti widgetektől kapott paraméterekkel, amely elvégzi a megfelelő módosításokat a gépeken, hogy törölje a kapcsolatot közöttük. Majd miután ez megtörtént itt is lehetőségünk van a log ablak megjelenítésére. Itt már láthatjuk, hogy miért volt célszerű a LogWindow ablakot megvalósítani úgy, hogy univerzálisan működjön hiszen minden ablak esetében feltudjuk használni hiába különböznek a logok, illetve a megjelenítők.

7.3.4. Controller réteg kialakítása

Az utolsó réteg, aminek a felépítését megtekintjük a Controller réteg lesz. Itt definiáljuk az üzletilogikát, ami nálunk főként Shell scripteket tartalmaznak. Ez a réteg fogja összekötni a másik két rétegünket. A View rétegtől kapott információkon műveleteket hajt végre majd azt továbbítja az adat réteg felé. Alapvetően mi két részre oszthatjuk ezt a réteget úgymint CreateTopologie, illetve ConfigureConnections. Már az elnevezésük is beszédes lehet. Az egyik a topológiát fogja kiépíteni, létrehozza az adott osztályokat majd feltölti azokat értékkel. A másik pedig az egyes csomópontokon fogja elvégezni a kívánt műveleteket. Mindezek megvalósításához szükség van egy alap konfigurációra, amely alapján a kód futni fog. Mielőtt végig haladnánk az egyes műveleteket szeretnék ezen végig haladni, hogy könnyebb legyen később az egyes metódusok megértése.

A kód tartalmaz egy *Batmanclasses.JSON* konfigurációs fájt, amit a felhasználónak a program futtatása előtt fel kell tölteni adatokkal. Az egyszerűbb megértés érdekében egy példán keresztül szeretném szemléltetni ennek módját:

7.17. ábra Batmanclasses.JSON fájl leírása

A fájl, mint ahogy a kiterjesztéséből is látható egy JSON fájl így ennek a szintaktikájának követi. A fájl alapból egy üres listát tartalmaz, ami csomópontokat jelképezi. A későbbiekben ezt a listát kell feltöltenünk majd tetszőleges számú elemmel. A Node-ok felépítése a következőképpen történik: minden Node rendelkezik az adatrétegben definiált tulajdonságokkal. Ehhez még hozzá adjuk az SSH kapcsolathoz tartozó IP címet, ami segít majd a későbbiekben az egyes gépek beazonosításában. Továbbá minden Node

rendelkezik egy taps nevezetű listával. Ez a lista fogja tartalmazni az egyes kapcsolatokhoz tartozó szükséges paramétereket. Azaz ennek a listának a tartalma alapján fogjuk az a *tap* osztály példányait létrehozni. A program használata során ügyelnünk kell arra, hogy mikor egy kapcsolatot beállítunk mindkét csomópont listájában szerepelni kell. Mivel az egyik oldalon kliens interfészként a másik oldalon pedig a szerver kapcsolódási pontjaként. A lista elemei tartalmaznak egy *type* – azaz típus – mezőt, ami a korábban említett „s”, illetve a „k” értéket hivatott. Amennyiben az utóbbit állítjuk be ügyelnünk kell arra, hogy adjuk meg a *remote* mezőt is, mivel ennek az IP címnek a segítségével fog kapcsolódni a kliens.

Emellett a fájl tartalmaz *Connections* nevű listát, ami majd az SSH kapcsolatok során lesz segítségünkre. Ennek elemei tartalmaznak egy IP címet, illetve felhasználó nevet, amivel majd SSH kapcsolatot létesít a program.

Elsőként a topológia beállítását végző műveleteket szeretném bemutatni. Mivel fontos, hogy a program elején már tisztában legyünk a topológiával hiszen a későbbiekben ennek segítségével szeretnénk műveleteket végezni. Egymásra épülés szempontjából legalulról fogjuk kezdeni azaz a *Topologia* osztály példányosításval.

```
#Create Topologie
def creatTopologie():
    cfg = config.config_from_json(str(fileBasePath / "Batmanclasses.json"), read_from_file=True)
    top = Topologie()
    ret = top.createTopologie() # network
    for i in range(0,len(cfg.nodes)):
       top.addNode(ret,cfg.nodes[i]["id"],cfg.nodes[i]["name"])

    for i in range(0, len(cfg.nodes)):
       for j in range(0,len(cfg.nodes[i]["taps"])):
            top.addEdge(ret,cfg.nodes[i]["taps"][j]["source"], cfg.nodes[i]["taps"][j]["destination"])

    top.generateGraph(ret)
    return top
```

7.18. ábra createTopologie() metódus leírása

Ez a metódus kettős célt szolgál mivel egyrészről létrehoz egy *topologia* osztály, amihez a későbbiekben hozzáadjuk a csomópontokat. Másik részről pedig a megjelenítéshez ez végzi el a szükséges műveleteket. Miután létrehoztuk a *topologia* osztályt meghívjuk annak *CreateTopologie()* metódusát, ami a megjelenítéshez egy Network entitást fog létrehozni. Ez után ehhez a példányhoz fogjuk hozzáadni a csomópontokat, illetve a kapcsolatokat. Ennek végrehajtásához meg kell nyissuk a JSON fájlt majd annak tartalmát *cfg* változóba mentjük. Majd egy ciklus segítségével bejárjuk a fájlban található Node elemeket. A lista minden egyes elemét az *addNode()* függvény segítségével hozzáadjuk a topológiánkhoz azok neveit és Id érékeit. Így a hálózatunk vizuálisan már tartalmazza a csomópontokat, de a kapcsolatokat nem. Ezt egy dupla ciklus segítségével oldjuk meg úgy, hogy az elsővel bejárjuk a Node listát majd a másodikkal a *taps* lista elemeit. Az egyes elemeken pedig meghíjuk az *addEdge*() metódust a megfelelő

paraméterekkel. Azért lényeges, hogy külön ciklusban hozzuk létre a kettőt mivel amikor létrehozunk egy Node-ot és annak beállítanánk kapcsolatait akkor egy olyan csomópontra fog hivatkozni az *addEgde() függvény,* ami még jelenleg nem létezik. Így a program hibát fog eredményezni. Ezzel a Network objektumunkat inicializáltuk, így a következő lépésben meghívjuk a *generateGraph()* metódust, aminek paraméterkénk átadjuk. Végezetül pedig függvényünk visszatérési értékeként megadjuk a topologia osztály létrehozott példányát.

A következő függvények a Node osztály példányosítását fogják végezni. Ehhez két metódust fogok használni. Az egyik több Node példány feldolgozására képes amíg a másik pedig egy csomópontot fog létrehozni. Mindkét függvénynek meglehetősen egyszerű a felépítése.

```
#Create Nodes
def createNodes(Topoligie):
        for i in range(0, len(cfg.nodes)):
                SNode= node(cfg.nodes[i]["id"],cfg.nodes[i]["BatmanIP"],cfg.nodes[i]["name"])
                Topoligie.Append(SNode)
        return Topoligie

def createSingleNode(id,BatmanIP,Name,ConnIP):
    SNode = node(id,BatmanIP,Name,ConnIP)
    return SNode
```

7.19. ábra createNodes() és createSingleNode() fügények leírása

A *createNodes()* függvény szolgálja majd több entitás létrehozását egyszerre. Ehhez paraméterként megadunk neki egy korábban már létrehozott *Topologia* osztályt, amihez majd hozzácsatoljuk a csomópontokat. Majd a hasonlóan a megjelenítéshez itt is végig megyünk egy ciklus segítségével a konfigurációs fájl Node listájának elemein. Egy ideiglenes Node példány létrehozásával majd a topologia osztály *Append()* függvényével hozzácsatoljuk azt. Végrehajtjuk ezt a lista minden egyes elemén majd a függvény visszatér a módosított topológia példánnyal.

A *createSingleNode()* függvény egészen hasonló csak itt egyből paraméterként kapjuk meg az aktuális Node elemeit, majd azt egy ideiglenes példányban eltároljuk végül pedig ezzel térünk vissza. Egy különbséget vehetünk észre, hogy itt megadunk még egy *ConnIP* paramétert, ami az SSH kapcsolat kiépítése során fogunk használni.

A következő entitás, amit létrehozunk – a legkisebb egységünk – a *tap* osztály példányosítása lesz. A hálózatunk ugye eddig úgy néz ki, hogy van egy alap topológia példányunk, amihez hozzáadtunk több csomópontot. Most a minden egyes csomóponthoz hozzá kell adnunk a kapcsolatait, azaz a *tap* példányokat.

Így a függvényünk paramétereként meg kell adjuk az előre összeállított *topologia* példányunkat. Amivel a módosítás után a függvény végén visszatérünk. A *tap* példányok létrehozása hasonlóképpen fog történni, mint a Nodeok esetében. Egy ciklus segítségével végig megyünk a konfigurációs fájl *Node* elemein majd egy újabb for segítségével

bejárjuk az adott csomópont *taps* listáját. Eddig egyszerű dolgunk volt mivel minden csomópont azonos adattagokkal rendelkezett. Jelen esetben viszont a *tap* példányoknál kétfajta típusról beszélhetünk úgymint kliens és szerver oldali interfész. Így a létrehozás során egyből meg kell vizsgáljuk, hogy a lista adott eleme melyik típusba tartozik.

```
#Create Tap interface
def createTap(Topologie):

        for i in range(0,len(cfg.nodes)):
            for j in range(0, len(cfg.nodes[i]["taps"])):
                if cfg.nodes[i]["taps"][j]["type"] == "s":
                    STap = tap(cfg.nodes[i]["taps"][j]["type"],
                               cfg.nodes[i]["taps"][j]["port"], cfg.nodes[i]["taps"][j]["source"],
                               cfg.nodes[i]["taps"][j]["destination"],cfg.nodes[i]["kulcs"])
                    Topologie.Nodes[i].neighbors.append(STap)
                    Topologie.Nodes[i].NOTap = Topologie.Nodes[i].NOTap + 1

                else:
                    STap = tap(cfg.nodes[i]["taps"][j]["type"], cfg.nodes[i]["taps"][j]["port"],
                               cfg.nodes[i]["taps"][j]["source"],
                               cfg.nodes[i]["taps"][j]["destination"],cfg.nodes[i]["kulcs"],
                               cfg.nodes[i]["taps"][j]["remote"] )
                    Topologie.Nodes[i].neighbors.append(STap)
                    Topologie.Nodes[i].NOTap = Topologie.Nodes[i].NOTap + 1

        return topologie
```

7.20. ábra createTap() függvény leírása

Ezért rendelkezik a konfigurációs fájlunk a *type* mezővel, mivel ennek segítségével könnyedén meghatározhatjuk ezt. Az elágazás két ága ez alapján kerül kialakításra. Amennyiben a szerver oldali interfészről beszélünk létrehozunk egy ideiglenes változót, aminek értékül az adjuk tap példányt. Majd ezt a változót hozzáadjuk az aktuális Node *neighbors* listájához. Emellett növeljük csomópont *NOTtap* változóját, ami így majd az interfészek számát fogja tárolni az adott csomóponton.

Amennyiben ugye nem szerver oldali interfész a lista aktuális eleme akkor automatikusan a program kliensként fogja kezelni. Aminek konfigurációja egészen hasonló, de a példány létrehozásánál kiegészül a már többször említett *remote* mezővel. Ami a kapcsolat másik oldalán lévő szerver gép IP címe lesz. Majd hozzáadjuk ezeket a csomópont *neighbor*s listájához majd növeljük a számláló értékét. Végezetül pedig visszatérünk a módosított topologia példánnyal, ami a függvény lefutása után az aktuális teljes hálózatot eltárolta.

A következő üzletilogika, amit bemutatnék a ConfigureConnections.py fájlban található függvények lesznek. Az itt szerepelő függvények rakják össze és hajtják végre a Shell utasításokat, amelyeket majd a gépeken fogunk futtatni. Mindezt SSH kapcsolaton keresztül hajtja végre. A kapcsolat létrehozásában, illetve parancsok lefuttatásához egy a külső Python könyvtárat fogok használni, amelynek a bemutatása már nem része a szakdolgozatomnak. A függvények ezen csoportjában találhatóak az ablakok gombjainak lenyomásakor meghívott függvények.

Elsőként a teljes hálózat konfigurálását végző *isntallOpenMesh()* függvényt tekintsük meg. A metódusnak paraméterként megadjuk az előre létrehozott topologia példányt. Meg kell adjuk, hogy hogy melyik cél csomóponton szeretnénk beállítani, mivel a programnak tudni kell, hogy melyik géppel hozta létre az SSH kapcsolatot. Egy időben csak egy gépet tudunk konfigurálni mivel az *ex* változó egy végre hajtási környezetet hoz létre, ahol definiálnunk kell az SSH kapcsolatot ahhoz, hogy parancsokat tudjunk futtatni. A metódus ezt is paraméteren keresztül fogja megkapni. Mivel a topologia osztály nem tárolja a *ConnIP* értéket így ezt a konfigurációs fájlból fogjuk kiolvasni. Annak érdekében, hogy megtaláljuk a megfelelő csomópontot egy ciklus segítségével végig megyünk a JSON fájlban található csomópontokon. Megvizsgáljuk, hogy *ConnIP* változó megegyezik-e a host paraméter értékével, amennyiben igen, végrehajtjuk az alábbi parancsokat. Ezek végrehajtását végző külső rendszer nem része szakdolgozatomnak így ennek szintaktikájára nem fogok kitérni, de a szakdolgozat mellékletében a kód elérhető lesz.

```
def installOpenMesh(topologie,host,ex):
    for i in range(0, len(cfg.nodes)):
        if cfg.nodes[i]["ConnIP"] == host:
            ex.sEx(["apt-get", "update"], f)
            ex.sEx(["apt-get", "-y","install", "batctl"],f)
            ex.sPutText(iRLib.strip_margin(CreateBatmanNetwork(topologie.Nodes[i].batmanIp))
                        ,systemDNetwork + BNFilename,f)
            ex.sEx(["chmod","644",systemDNetwork+BNFilename],f)
            ex.sPutText(iRLib.strip_margin(batmanModules),moduleLoad + modulesConfig,f)
            ex.sEx(["apt-get","install","-y","openvpn"],f)
            for j in range(0,len(topologie.Nodes[i].neighbors)):
                if topologie.Nodes[i].neighbors[j].type == "s":
                    ex.sPutText(iRLib.strip_margin(CreateTapconf(cfg.nodes[i]["taps"][j]["id"],
                                                                 topologie.Nodes[i].neighbors[j].type,
                                                                 topologie.Nodes[i].neighbors[j].port)),
                                openVpn + tapName + cfg.nodes[i]["taps"][j]["id"] + ".conf",f)
                else:
                    ex.sPutText(
                        iRLib.strip_margin(CreateTapconf(cfg.nodes[i]["taps"][j]["id"],
                                                         topologie.Nodes[i].neighbors[j].type,
                                                         topologie.Nodes[i].neighbors[j].port,
                                                         topologie.Nodes[i].neighbors[j].remoteIP)),
                                openVpn + tapName + cfg.nodes[i]["taps"][j]["id"] + ".conf", f)
            ex.sPutText(iRLib.strip_margin(cfg.nodes[i]["kulcs"]), openVpn + tapName + ".key", f)
```

7.21. ábra installOpenMesh() függvény leírása 1

Két féle beépített függvényt fogunk használni az egyik *sEx()* mikor egy Shell parancsot akarunk lefuttatni. A másik amikor egy fájlba szeretnénk beleírni, ekkor az *sPutText()* metódust hívjuk meg a környezeten. Itt meg kell adjuk a fájl tartalmát string formátumban, illetve a fájl elérési útját a gépen.

Először, lefuttatjuk **apt-get update** parancsot, hogy a belső repositoryban található csomagok a legfrissebb verzióval rendelkezzenek. Majd ehhez hasonlóan letöltjük a batctl csomagot, ami batman-adv útválasztási protokolt tartalmazza. Amint a telepítés

megtörtént elkészítjük a bat0 konfigurációs fájlt a CreateBatmanNetwork() metódus segítségével, aminek paramétereként megadjuk a topologia aktuális csomópontjának *batmanIP* értékét. Az összeállított fájl kinézetére példát a tesztkörnyezet kialakítása című fejezetbe találhatnak. A létrehozás után megváltoztatjuk ennek a fájlnak a jogosultságait, hogy futtatható legyen a későbbiekben. A jogosultságok módosítása után be kell töltenünk a batman-adv modult, hogy a gép újra indítása után egyből automatikusan elinduljon. Következő lépésben telepítjük az OpenVPN szolgáltatást eddigiekhez hasonlóan. Majd elkezdjük a tap interfészek konfigurálását. Ezt egy ciklus segítségével tesszük meg ami végig megy a Node osztály *neighbors* listájának elemein. Itt az eddigiekhez hasonlóan különbséget kell tennünk a szerver és a kliens interfészek között. Ezt egy logikai elágazás segítségével hajtjuk végre, ami a *type* mező értékét vizsgálja az adott csomóponton. Majd ennek megfelelően kerül a konfigurációs fájl összerakásra a CreateTapConf() függvény segítségével. Ez a metódus egy string értékkel fog visszatérni, amit bemeneti paraméterekből megformázva és összefűzve fog visszaadni. Miután a tap interfészek elkészültek létrehozzuk hozzá a statikus kulcsot.

```
ex.sEx(["systemctl", "enable", "openvpn"], f)
ex.sEx([openVpnService, "restart"], f)
ex.sEx(["systemctl","restart","systemd-networkd"],f)
ex.sPutText(iRLib.strip_margin(waitSH(cfg.nodes[i]["taps"][0]["id"])), "/root/script.sh", f)
ex.sEx(["sh","script.sh"],f)
for j in range(0,len(topologie.Nodes[i].neighbors)):
    ex.sEx(["batctl", "if", "add", tapName + cfg.nodes[i]["taps"][j]["id"]], f)
ex.sEx(["ip", "link" ,"set" ,"up", "dev","bat0"],f)
for j in range(0, len(topologie.Nodes[i].neighbors)):
    ex.sEx(["ip", "link", "set", "up", tapName + cfg.nodes[i]["taps"][j]["id"]], f)
```

7.22. ábra installOpenMesh() függvény leírása 2

A működéshez szükséges összes fájlt létrehoztuk így ez után következik, hogy az OpenVPN státuszát engedélyezzük majd újra indítjuk, hogy létrehozza az újonnan hozzáadott interfészeket a megadott fájlok alapján. Majd újra indítjuk a systemD-networkD hálózati vezérlőt, hogy létrehozza a bat0 interfészt. Végső lépésünk pedig az lenne, hogy Batman hálózathoz hozzáadjuk az újonnan hozzáadott tap interfészeket és felkapcsoljuk azokat.

Viszont a fejlesztés során egy olyan probléma lépett fel, hogy az újraindítás közben az interfészek létrehozása több időt vesz igénybe. Így amikor a program tap interfészt próbája felkapcsolni az még nem létezik és hibába ütközik. Ennek megoldására egy bash scriptet készítettem, ami többek között egy while ciklust tartalmaz, ami egészen addig nem lép tovább ameddig az adott interfész meg nem jelenik. Amint az interfész megjelenik a program tovább léphet és egy ciklus segítségével felkapcsolhatja az összes tap interfészt. Ennek a függvénynek a meghívása egy *factory()* metódusban történik, ahol először is kiürítjük a log fájlokat, majd létrehozzuk a topológia osztály a JSON fájl

alapján, úgy hogy egy változóba értékének megadjuk a *CreateTopologie.Generate()* függvény visszatérési értékét. Majd a JSON fájlban található Connections listát bejárjuk egy ciklus segítségével. A cikluson belül létrehozunk egy *tConn* változót, ami az SSH kapcsolatot fogja létrehozni. Ezt majd későbbiekben az *Executernek* fogjuk megadni. A kapcsolat létrehozásához a Connections lista elemeinek *ConnIP* címét, illetve a hozzátartozó *user* nevet fogjuk felhasználni. Végül pedig meghívjuk az *installOpenMesh()* függvényt az előbb létrehozott connection, illetve topológia objektummal. Amennyiben a függvény hiba nélkül lefutott True értékkel fog visszatérni.

A következő metódus, aminek működését bemutatom egy független csomópont hozzáadásáért lesz felelős. Ennek felépítése hasonló lesz, mint az előbb bemutatott kódsor. Ebben az esetben nem kell a konfigurációs fájlból kikeresni az adott csomópontot, mivel a függvény bemeneti paramétereként kapjuk meg a beállításhoz szükséges adatokat. Így a programok telepítését, illetve konfigurációs fájlok létrehozását nem részletezem. Ugyanakkor két lényegi részben tér el ez a metódus az előzőtől. Az első mikor egy kapcsolatot szeretnénk beállítani ahhoz mind a kliens mind a szerver gépen be kell állítanunk a megfelelő interfészeket. Ennek magvalósítására létrehoztam egy kliens, illetve egy szerver oldali függvényt. Ezeknek felépítése közel ugyanolyan csupán a tap interfész konfigurációja különbözik. Majd ez a két függvényt egy factory tervezési mintához hasonlóan meghívjuk a felhasználói bementekkel. A második különbség amikor a teljes hálózatot állítjuk be a JSON fájlból csak kiolvassuk az adatokat, de jelen esetben a topológiánk bővül így ezt hozzá kell adnunk a konfigurációs fájlunkhoz. Ennek megvalósítására a *writeJSON()* függvény szolgál:

```
def writeJSON(conn,serverIP,port,RemoteIP):
    with open(filename) as fp:
        listObj = json.load(fp)
    SUMNodeId=len(listObj["nodes"])+1
    NodeExist =False
    for i in listObj["nodes"]:
        if i["ConnIP"] == conn:
            i["taps"].append({"id":str(len(i["taps"])+1),"type":"k"
                              ,"source":FindNodeId(conn,listObj)
                              ,"destination":FindNodeId(RemoteIP,listObj)
                                      ,"port":port,"remote":FindVPC(RemoteIP) })
            NodeExist=True
    if NodeExist == False:
        listObj["nodes"].append({"id":str(SUMNodeId),"name":"Test "+str(SUMNodeId),
                            "BatmanIP": serverIP, "ConnIP": conn,
                            "taps":[{"id":"1","type":"k","source":str(SUMNodeId)
                                      ,"destination":FindNodeId(RemoteIP,listObj)
                                      ,"port":port,"remote":FindVPC(RemoteIP) }],
                            "kulcs": listObj["nodes"][0]["kulcs"]
                             })
    with open(filename, 'w') as json_file:
        json.dump(listObj, json_file,
                  indent=4,
                  separators=(',', ': '))
```

7.23. ábra writeJSON() függvény leírása

A függvénye paramétereiként megadjuk a kapcsolat beállításaihoz használt felhasználó által beírt adatokat. Bemenetként megkapjuk az SSH kapcsolódáshoz használt kliens és server IP címet, a tap interfészben használt port számát, illetve a Batman hálózaton használt IP címet. Első lépésként megnyitjuk a fájl olvasásra, majd annak tartalmát betöltjük a *listObj* változóba. Ilyenkor a fordító környezet automatikusan szerializálja fájl tartalmát és objektumokba rendezi nekünk. A későbbi használathoz elmentjük egy külön változóba a csomópontok lista hosszát, hiszen ennek segítségével határozzuk meg az új Node id értékét. Majd létrehozunk egy logikai változót, aminek értéke fogja eldönteni, hogy az adott csomópont létezik-e már a hálózaton. Ahhoz, hogy ennek értéket adjuk végig kell iterálnunk *nodes* lista elemein, majd megvizsgáljuk, hogy a bemenetként kapott kapcsolódási IP megegyezik-e csomópontok valamelyikével. Amennyiben szerepel csak a *taps* interfész listához kell hozzáadnunk a megfelelő konfigurációt. Különösen figyelnünk kell, hogy a remote mezőben megadott IP nem egyezhet a SSH kapcsolódáshoz használt IP címmel, mivel ezen keresztül a gépek nem tudnak kommunikálni. Így a gépek VPC hálózaton használt címét fogjuk kikeresni a konfigurációs fájlból *FindVPC()* függvény segítségével. Majd a logikai változónk értékét True-ra állítjuk mivel a csomópont már létezik a listában. Amennyiben a *NodeExist* logikai változó értéke nem változik a csomópontot is hozzá kell adnunk a változóhoz a tap interfész mellett.

Miután megvagyunk az objektumok hozzáadásával a listához, vissza kell írnunk a listObj változó értékét a JSONJSON fájlba. Ehhez megnyitjuk az eredeti fájlt „w” paraméterrel, azaz írás művelethez majd a *dump()* beépített függvény segítségével beleírjuk a fájlba a megadott objektumot, illetve megadjuk a szintaktikához használt írásjeleket és a megfelelő indentálást. Ezzel rögzítve az új csomópontot a konfigurációs fájlunkba. Mivel a kapcsolatot nem elég csak a kliens oldalon létrehoznunk így a server konfiguráláshoz egy ehhez hasonló függvényt fogunk használni. A bemeneti értékekben, illetve a tap interfész remote mezőjében tér majd el mivel nem kell azt tartalmaznia, de a kód további felépítése azonos lesz.

Az utolsó függvény, aminek működésén végig haladunk a csomópontok törléséért lesz felelős. A metódus több dologban is eltér az eddigiektől hiszen eddig csak kapcsolatokat hoztunk létre, de törölni nem töröltük azokat. A függvény paramétereként megkapjuk a törlendő csomópont SSH kapcsolathoz használt IP címét és a törölni kívánt tap interfész portját, illetve a Executer környezetet a Shell parancsok futtatásához. A metódus elején megnyitjuk a JSON konfigurációs fájlunkat mivel onnan is ki kell törölnünk az adott kapcsolatot. Először végig iterálunk a *nodes* listán és ahol egyezik *ConnIP* mező a bementként kapott IP címmel azon a csomóponton kell a parancsokat futtatnunk. Majd a kiválasztott Node-on *taps* listáján végig haladunk és megvizsgáljuk melyik port egyezik a bemeneti paraméterrel. Amint megtaláltuk a törölni kívánt kapcsolatot lefuttatjuk az **ip link delete tap$id** parancsot az id helyére pedig az aktuális tap interfész id értékét. Ezzel törölve a kapcsolatot majd, annak érdekében hogy kitöröljük a tap konfigurációját

ismeghívjuk az **rm** parancsot a fájl nevével.Miután megszakítottuk a kapcsolatot a JSON fájlból is ki kell töröljük.. Itt ahelyett, hogy a JSON fájt elemeit járnánk be, a szerializált objektumokon fogunk végig menni. Hasonló módon az előbbi ciklushoz megkeressük a törölni kívánt kapcsolatot. Majd meghívjuk a *taps* lista aktuális elemén pop beépített metódust ezzel kitörölve a lista kívánt elemét. A törlés után pedig a **break** parancsot hívjuk meg. Amennyiben ezt nem tennénk a program hibába ütközne mivel a ciklus a lista hosszáig megy, aminek az elemét egyel csökkentettük így a feltétel vizsgálat során egy olyan indexen lévő elemet vizsgálunk, amely már nem létezik.

```
def DeleteConnection(sourceIP,port,ex):
    with open(filename) as fp:
        listObj = json.load(fp)
    for i in range(0, len(cfg.nodes)):
        if cfg.nodes[i]["ConnIP"] == sourceIP:
            for j in range(0,len(cfg.nodes[i]["taps"])):
             if cfg.nodes[i]["taps"][j]["port"]==port:
                 ex.sEx(["ip", "link", "delete","tap" + cfg.nodes[i]["taps"][j]["id"]], f)
                 ex.sEx(["rm", openVpn +"tap" + cfg.nodes[i]["taps"][j]["id"] + ".conf"], f)

    for i in listObj["nodes"]:
        if i["ConnIP"] == sourceIP:
            for j in range(0, len(i["taps"])):
                if i["taps"][j]["port"] == port:
                    i["taps"].pop(j)
                    break
    with open(filename, 'w') as json_file:
        json.dump(listObj, json_file,
                  indent=4,
                  separators=(',', ': '))
```

7.24. ábra DeleteConnection() függvény leírása

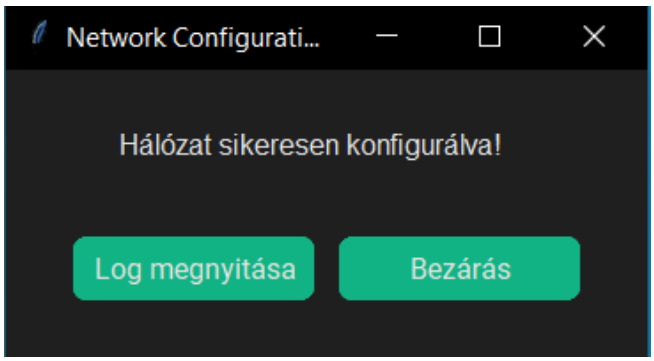
Végezetül visszaírjuk a módosított *listObj* változó tartalmát a JSON fájlba a már előbb bemutatott módon. Hasonlóan a szimpla csomópont hozzáadása függvényhez ezt is egy factory mintához hasonlóan hívjuk meg, mivel végre kell hajtanunk a szerver és a kliens gépen is.

7.3.5. Kód futtatása

Az előző fejezetben részletes betekintést kaptunk a kód lényegesebb elemiről, végig haladtunk az általános felépítésén. Következő lépésünk, hogy az elméleti síkból kimozdítva futtassuk az általunk megírt kódot és megfigyeljük működését. Ahhoz, hogy ez véghez vigyük el kell készítenünk a kívánt rendszer JSON konfigurációs fájlját. Mivel a rendszer felépítéséből fakadóan roppant rugalmas a mesh kialakításnak köszönhetően így idő közben könnyedén tudunk változtatni rajta. Rendszerünk alapjául 2 szerver gép fog szolgálni, mivel ezek valószínűleg nagyobb forgalmat fognak kezelni így erősebb specifikációt választottam. Az egyik gép 8vCPU-t, illetve 32 GB ram-ot kapott míg a másik 4vCPU és 8GB rammal rendelkezik majd. Jelen esetünkben 3 kliens eszközről

beszélhetünk a rendszer indításának kezdetén. A kód futtatása előtt előre definiálhatjuk a konfigurációs fájlban a kapcsolatainkat vagy a rendszer futása közben is beállíthatjuk azokat. Én az első megoldást fogom választani. Így a fájl feltöltésével kezdem a program futtatása előtt. Elsőként meg kell határoznunk a csomópontok számát, ami jelen esetben 5 darab lesz. Mindegyik Node-nak adnunk kell egy Batman hálózaton használt IP címet emellett beállítjuk a konfigurációhoz szükséges floating IP-t illetve a VPC használt IP címet is. A következő és roppant lényeges szempont a tap interfészek beállítása lesz. Ezt csak felületesen fogom érinteni mivel a tesztelés fejezetben részletesebben foglalkozunk vele, emellett a JSON fájl megtalálható a mellékletek között. A kapcsolatok kiépítésekor próbáltam figyelembe venni, hogy melyik eszközön milyen terhelés várható. Így a két szerver gép között 2 odavissza kapcsolat is van, ami a túlzott forgalmat hivatott kezelni. Fontos megjegyezni, hogy Batman hálózat két csomópont közötti útvonalakat nem aggregálja külön-külön kezeli őket. Az első kliensünk mindkét szerverrel kapcsolatban áll. Tesztelés szempontjából az egyik tap interfész esetében ő maga is szerverként funkcionál. A következő kliensünk a kettes id-val rendelkező kizárólag a hármas számú klienssel áll kapcsolatban. Az utolsó hármas gépünk pedig kettes számú szerverrel fog kommunikálni. Így elérve azt, hogy a kettes kliens gép úgymond a hálózat peremén van és kizárólag a hármas gépen keresztül tud kommunikálni.

Miután elkészünk a konfigurációs fájlunk futtathatjuk is a programotMost pedig, hogy ellenőrizzük a JSON fájlunk helyességét nyomjunk rá a topológia megjelenítése gombra. A program a háttérben meghívja a szükséges függvényeket majd megjeleníti a megfelelő ablakot abban az általunk leírt topológiával. Ez a *7.12* -es ábrán tekinthető meg. Miután a gráf megjelent azt bezárva kattintsunk a hálózat konfigurálása gombra, hogy kiépítsük az összeállított rendszert. A rendszer pár perc után ezzel az ablakkal tér vissza, hogy hálózat sikeresen konfigurálva.



7.25. ábra Hálózat sikeresen konfigurálva ablak

Itt lehetőségünk van a naplófájlok megnyitására, így megnézzük, hogy minden az elvártak szerint történt-e. A log nem tartalmazott hibát így a rendszer hibamentesen elindult. A program, illetve a rendszer tesztelését a következő 8. fejezetben mutatom be.

8. Rendszer tesztelése és futtatása

A kód futtatása után a JSON fájlban definiált rendszer elkészünk és használatra kész. Mivel a kód a leírtak alapján lefutott ez az jelenti, hogy szintaktikai szinten megfelelt, illetve a Shell parancsok sem ütköztek hibába. Viszont a kapcsolatok logikai szerkezetét nem ellenőrzi a rendszer így ezeket nekünk kell megvizsgálnunk. Emellett a rendszer alapvető célját is a végponttól-végpontig terjedő kommunikációt is szükséges tesztelnünk. Továbbá szeretném az egyes funkciók felhasználói felületről vezérelhető funkciók segítségével tesztelni a rendszer, hogy reagál egyes kapcsolati meghibásokra.

A tesztelés során egy kiemelt csomópont szemszögéből fogok végig haladni az egyes teszteseteken, Ez a csomópont az egyes számú kliens lesz mivel minden tesztesethez megfelelően helyezkedik el a hálózatban. Elsőként, hogy megbizonyosodjunk, hogy a virtuális gép kapcsolatai sikeresen kiépültek és részese a Batman hálózatnak futtatam a *batctl n* parancsot:

```
root@OpenVPN-kliens001:/etc/openvpn# batctl n
[B.A.T.M.A.N. adv 2019.4, MainIF/MAC: tap1/c2:17:1b:1c:2c:73 (bat0/c2:af:dc:62:2f:c9 BATMAN_IV)]
IF             Neighbor              last-seen
        tap1     a2:32:94:5c:95:8d    0.608s
        tap2     4e:f4:42:dd:22:f6    0.204s
```

8.1. ábra Klines 1 gép Batman szomszédai

Ahogy azt vártuk a kiválasztott gépnek pontosan 2 közvetlen szomszédja van Batman hálózaton, amikkel a tap1, illetve a tap2 interfészeken kommunikál. Ezzel megbizonyosodtunk arról, hogy a gép részese a Batman hálózatnak, így a következő teszt a rendszeren, hogy a két legtávolabb lévő eszköz tud-e egymással kommunikálni.

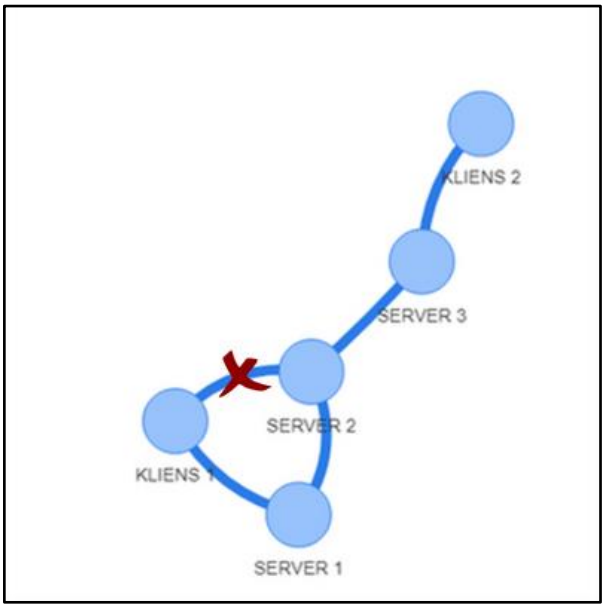
Ehhez kiválasztjuk a 2 számú kliens gépet, a kommunikáció tesztelésére szükségünk lesz a Batman hálózaton használt 10.10.10.22 IP címre hiszen ezen keresztül próbálunk majd vele kapcsolatot teremteni. Ehhez az egyes gépről fogjuk futtatni a *batctl traceroute 10.10.10.22* parancsot:

```
root@OpenVPN-kliens001:/etc/openvpn# batctl traceroute 10.10.10.22
traceroute to 10.10.10.22 (de:c0:f2:a8:f9:cc), 50 hops max, 20 byte packets
 1: 56:c1:76:fd:f1:fa  0.764 ms  0.721 ms  0.713 ms
 2: fe:78:9d:ce:1f:d9  0.885 ms  0.838 ms  0.831 ms
 3: de:c0:f2:a8:f9:cc  1.001 ms  0.956 ms  0.963 ms
root@OpenVPN-kliens001:/etc/openvpn#
```

8.2. ábra Végpontok közötti teszt 1 eset

Ahogy látjuk a képen a parancs sikeresen lefutott és a két gép között kiépült a kapcsolat. A traceroute parancs sajátossága, hogy segítségével követhetjük a csomag útvonalát. Egy hagyományos 3. rétegen futtattot traceroute parancs IP címeit fogja kiírni az egyes állómásoknak de mivel a Batman protokoll szigorúan második rétegeken közli az adatokat így Mac címeket fogunk látni. A csomagnak 3 állomáson keresztül kellett mennie ahhoz, hogy megérkezzen a céleszközre. Ami a mi esetünkben igazolja a hálózat helyes kiépítését mivel a két csomópont 3 ugrásnyi távolságra van egymástól. Ennek a tesztnek eredményeképpen bebizonyítottuk, hogy a hálózat bármely tagja képes

kommunikálni az összes többivel. A következő teszteset egy kapcsolat elvesztése lesz. Mivel a rendszerünket úgy terveztük, hogy amennyiben van lehetősége kapcsolat vesztés esetén új útvonalat keressen így először kitörlünk egy útvonalat, majd megfigyeljük az útvonal alakulását.



8.3. ábra kapcsolat törlésének szemléltetése

Ennek kivitelezésére a program „Csomópont leválasztása” funkcióját fogom használni, ami két gép közötti tap interfészen keresztüli kapcsolatot hivatott törölni. Ezzel egyszerre tudjuk tesztelni a program működését a rendszer viselkedésével együtt. Ehhez először megvizsgáljuk, hogy alapesetben merre indul a csomag majd azt az útvonalat kitöröljük. Ahogy azt az előző tesztesetből láthattuk a csomag az egyes kliens felől a kettes szerver gép felé indult. Így ezt az útvonalat töröljük ki. Ehhez a főmenüből meghívott törlés ablaknak megadjuk a két gép floating IP címét, illetve a közöttük húzódó kapcsolatnak a portját. Majd rányomunk a „kapcsolat törlése” gombra ezzel kiadva a parancsot a rendszernek a módosításra. A sikeres lefutás után, meghívjuk az előző traceroute parancsot, hogy lássuk van e változás állt be az útvonalban.

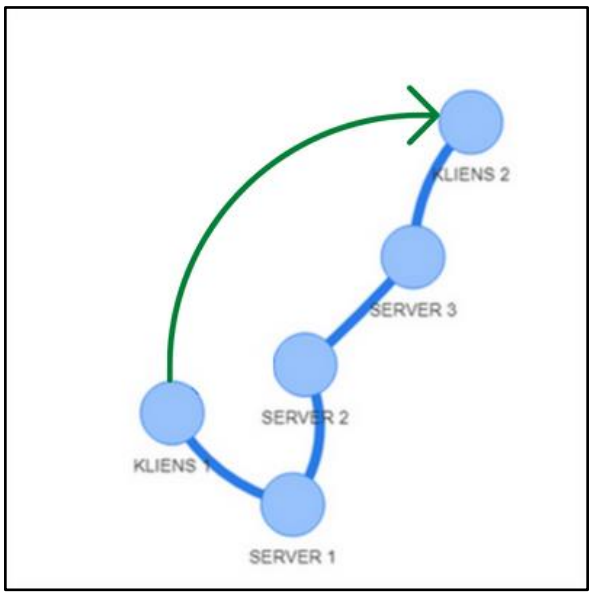
```
root@OpenVPN-kliens001:/etc/openvpn# batctl traceroute 10.10.10.22
traceroute to 10.10.10.22 (de:c0:f2:a8:f9:cc), 50 hops max, 20 byte packets
 1: 9e:7d:9e:30:a3:c0  0.721 ms  0.611 ms  0.609 ms
 2: 56:c1:76:fd:f1:fa  0.826 ms  0.767 ms  0.781 ms
 3: fe:78:9d:ce:1f:d9  0.898 ms  0.882 ms  0.902 ms
 4: de:c0:f2:a8:f9:cc  1.024 ms  1.011 ms  0.984 ms
```

8.4. ábra Végpontok közötti teszt 2 eset

Ahogy látható a 8.4 képen a csomag útvonala megváltozott. Az eddigi három állomás helyett most négy csomóponton keresztül épült fel a kapcsolat. Ami bizonyítja, hogy a rendszer képes reagálni a kapcsolat megszakadásra mivel egyből alternatív útvonalat keresett. A törlés után a Batman protokoll az egyes szerveren keresztül érte el a cél állomást, ennek következtében növekedett meg az állomások száma négyre. Emellett

igazoltuk, hogy a kapcsolat törlése funkció is helyesen működik mivel a kívánt útvonal megszűnt.

A következő teszt során azt vizsgáljuk, hogy egy csomópont kiesése után, amennyiben hozzáadunk egy új útvonalat, képes-e alkalmazkodni a rendszer és az új útvonal segítségével helyreállítani a kapcsolatot a gépek között. Ehhez a hasonló parancsot fogjuk futtatni, mint eddig de most az egyik központi szerver gépünket kikapcsoljuk. Ezzel szimulálva, hogy az egyik csomópontunk váratlanul kiesik a topológiából. A legegyszerűbb megoldás, ha a felhőszolgáltatón keresztül egy gombnyomással kikapcsoljuk gépet. Majd ez után lefuttatjuk az eddig használt traceroute utasítást. A parancs eredményeként először a csomag eljut az egyes szerverre majd elakad mivel nem tud tovább haladni.



8.5. ábra Kapcsolat hozzáadásának szemléltetése

Ennek megoldására a program segítségével egy alternatív útvonalat fogok létrehozni, hogy a kommunikációt visszaállítsuk. Ehhez meghívjuk az „Új csomópont hozzáadása funkciót”, ahol megadjuk a kliens és szerver adatokat. Jelen esetben az egyes, illetve a kettes kliens gép között szeretnénk kapcsolatot kiépíteni. Így megadjuk ezek floating IP címét, az egyes gép Batman címét, tap interfész portját. Majd „Csatlakozás” gomb megnyomásával a program felkonfigurálja az kért kommunikációs csatornát. Ahhoz, hogy a hálózat két legtávolabbi eleme között a kapcsolatot teszteljük az egyes, illetve a hármas szerver kommunikációját fogjuk vizsgálni. Az előző esethez képes azért volt szükséges a végpontok módosítása mert az egyes és a kettes gép szomszédokká váltak.

```
root@OpenVPN-kliens001:~# batctl traceroute 10.10.10.20
traceroute to 10.10.10.20 (fe:78:9d:ce:1f:d9), 50 hops max, 20 byte packets
 1: a2:32:94:5c:95:8d  0.781 ms  0.733 ms  0.735 ms
 2: fe:78:9d:ce:1f:d9  0.940 ms  0.897 ms  0.858 ms
```

8.6. ábra Végpontok közötti teszt 3 eset

Így a traceroute parancsot meghívjuk a hármas gép IP címével. Azt tapasztaljuk, hogy a csomag két állomáson keresztül haladt majd elérte a kívánt célt. Ebből következik, hogy a rendszer adaptálta az új útvonalat és azt használva képes volt kommunikálni a hálózat két legtávolabbi pontja között. Emellett megbizonyosodtunk arról is, hogy a programunk csomópont hozzáadása funkciója hiba nélkül működik.

Ezzel a rendszer tesztelését lefedtünk az általunk állított szempontok alapján, mivel megvizsgáltuk, hogy hogy reagál egyes meghibásodásokra, illetve új kapcsolat beépítésére. A fő szempontunk a tesztelés során az volt, hogy a rendszer mindvégig rugalmas maradjon és képes legyen alkalmazkodni. Ennek kritériumnak pedig teljes mértékben a megvalósított hálózat.

9. Továbbfejlesztési lehetőségek

Az informatika, mint iparág elképesztő módon növekszik, mindig adódnak új területek és ötletek, így az általam fejlesztett program is rendelkezik a jövőbeli tovább fejlődés lehetőségével. Már a rendszer létrehozásánál fontos felmérni, , hogy milyen irányba tud tovább fejlődni . Mivel a dolgozatomban használt technológiák mind nyílt forráskóddal rendelkeznek, így ezek is várhatóan nagy mértékben fognak változni. Emiatt az általam összerakott rendszer számos tovább fejlesztési lehetőséggel rendelkezik.

Az egyik jelentős előre lépési lehetőség az lehet, hogy várhatóan a linux kernel részévé válik az OpenVPN. Ennek során lehetőségünk lesz kiterjeszteni a szolgáltatásunkat olyan további eszközökre, amelyek linux rendszermaggal rendelkeznek. Így például , egyes okostelefonok a jövőben átvehetik a kliens gép szerepét.

További lehetőség lehet rendszerünk fejlesztésére, hogy kényelmesebbé tesszük a felhasználói élményt. Ilyen átalakítás például, hogy a Topologia megjelenítéséhez használt ablakot további funkciókkal bővítjük. Hasznos a felhasználó számára, ha ábrázoljuk , hogy az egyes csomópontok között hány kapcsolat húzódik. Emellett a program mérné, hogy a kapcsolódó tap interfészek között milyen sebességű a csomag átvitel és ezt a képernyőn is jelezné.

A program egyik nehézége, hogy kézzel kell összeállítani a JSON fájl tartalmát. Ez meglehetősen bonyolult és nagy eséllyel kerül bele emberi hiba. Ennek kiküszöbölésére megoldás lehet egy olyan felhasználó barát felület, ahol grafikusan összerakható a konfigurációs fájl. A program futása elött egyből ebben a szerkesztőben indulna a program. Itt dinamikusan lehetne állítani a csomópontok számát, illetve összehúzni azokat egy VPN kapcsolattal. Ezzel sokkal kisebb lenne az emberi hiba valószínűsége, illetve a program felkonfigurálásának ideje is lerövidülne.

A program hátrányai közé tartozik a hosszú futásidő a bonyolultabb műveletek esetén, ilyen például a hálózat teljes felkonfigurálása. Mivel a program egyszerre csak egy gépet tud beállítani, így végig kell várni ameddig befejezi annak konfigurálását és csak utána tud átlépni a következő gépre. Nagy mértékben gyorsítaná a függvények végrehajtását, ha több szálon tudna futni párhuzamosan egyszerre a konfigurálás. Mivel az egyes gépek a beállítás során nem függnek egymástól, lehetőségünk lenne azokat egy időben konfigurálni. A parhuzamos végrehajtás nyilvánvalóan sebesség növekedést eredményezne, ugyanakkor a program erőforrás igénye is megnőne. Az ismertetett probléma felveti a kérdést, hogy a felhasználónak melyik szempont fontosabb a program futtatása során, amit a fejlesztéskor figyelembe kell venni.

10. Összefoglalás

A szakdolgozatom célja volt, hogy megoldást találjak egy olyan problémára, amely képes reagálni a hálózati változásokra, valamint képes alkalmazkodni az otthonokban kialakult folyamatosan változó környezethez. Egy ilyen rendszer kialakítása számos környezetben nyújthat megoldást például, amit a korona vírus idézett elő az életünkben. Ennek érdekében az első félévben kutatómunkát végeztem az irodalomban alkalmasnak vélt technológiák működéséről. Majd ezeket összevetettem, hogy a megvizsgált irodalom alapján ki tudjam választani a megfelelőt a problémára. Először is rendszerünk távoli elérésének érdekében kiválasztottam OpenVPN community megvalósítását. A fő szempont a technológia kiválasztásnál az volt, hogy egy nyílt forráskódú, teljesen ingyenes projekt legyen, emellett nagy előnye, hogy hamarosan a linux kernel része lesz. Az útválasztó protokollnak a batman-adv-t választottam, mivel mesh hálózatokhoz lett tervezve, ami a mi megvalósításunk alapja lesz. A részletes szempontlistát és kiértékelés a dolgozatom megfelelő fejezete tartalmazza.

Ezek után létrehoztam egy teszt környezetet virtuális gépek segítségével, ami képes volt szimulálni a végleges rendszert. Megvizsgáltam, hogy reagálna a rendszer egy esetleges csomópont vesztésre. Majd levontam a konzekvenciákat és előkészítettem egy rendszer tervet az éles rendszernek, ami képes megoldani a korábban felvezetett problémát.

A szakdolgozat második része ennek a tervnek a megvalósítását öleli fel. A tesztkörnyezetből levont tanulságokat figyelembe véve alakítottam ki a rendszert. A kötöttebb, lokális csomópont helyett, felhő alapú megvalósításra váltottam. Mivel egy rugalmasabb és könnyeben menedzselhető környezetre volt szükségem.

Annak érdekében, hogy minél egyszerűbb legyen a csomópontok konfigurációja egy Python scriptet készítettem az alkalmazás konfigurálására és beállítására. A script MVC tervezési minta alapján készült annak érdekében, hogy a program áttekinthetővé váljon, illetve az egyes rétegek jobban elkülönüljenek egymástól. A program fejlesztése során több nehézséggel is szembe kellett néznem, melyek a munka során igyekeztem megmagyarázni és áthidalni. A konfigurálási folyamatokat egy külső menedzser gép segítségével akartam vezényelni, így az egyes gépeket SSH kapcsolaton keresztül állítottam be.

Abból a célból, hogy a gépek beállítása felhasználóbarát legyen , szükségem volt egy megjelenítési felületre. Erről a grafikus interfészről lehetett a későbbiekben csomópontot hozzáadni és törölni. A program képes megjeleníteni az aktuális topológiát gráfok segítségével, illetve a hálózat összes gépét beállíthatjuk az általunk meghatározottak szerint. A program futtatása előtt szükséges, hogy előre definiáljuk a létrehozni kívánt tap interfészeket, illetve a csomópontokat az erre használt JSON fájlban.

A topológia a megfelelő utasítások meghívása után egészen rövid időn belül konfigurálásra került. Majd ennek a rendszernek a tesztelésének érdekében végeztem méréseket. Kiválasztottam egy adott kliens csomópontot és annak a „nézőpontjából” vizsgáltam az adott teszteseteket. Az esetekkel megvizsgáltam a hálózati problémákat, valamint a rendszer validálását is elvégeztem.

Elsőként megvizsgáltam, hogy a kliens tud-e kommunikálni szomszédaival, majd a legtávolabb lévő csomóponttal is képes-e kapcsolatot teremteni. Ezután kitöröltem a

topológiából egy VPN kapcsolatot, majd később lekapcsoltam egy csomópontot, hogy lássam miként reagál a rendszer az esetleges hálózati nehézségekre. A program véglegesítése után megvizsgáltam, hogy milyen lehetősége van a fejlődésre kreált szoftvernek, illetve az architektúrának. Az OpenVPN nyílt forráskódjából és a Linux kernelbe építéséből fakadóan hatalmas potenciál maradt benne. Emellett a program futtatásához szükséges JSON fájl egyszerűsítését is további céljaim közé tűztem ki.

A dolgozatom összegzésként elmondható, hogy sikerült kialakítani egy olyan távolról elérhető komplex rendszert, amely képes kezelni az esetleges hálózati nehézségeket is. Ez a konfiguráció a terheléstől függően képes a fel-le skálázhatóságra igény alapján, illetve a magas rendelkezésre állásra. Tanulmányom során, sikerült egy meglehetősen bonyolult architektúrát egy egyszerű felhasználó felület által vezérelhetővé tenni.

11. Conclusion

The aim of my thesis was to find a solution to a problem created by the situation of Coronavirus. I wanted to design a system that could adapt to network changes and adapt to the ever-changing environment at home. To achieve this, I spent the last semester overviewing technologies that I thought were suitable for addressing the problem statement. Then compared these to the literature reviewed and selected the most appropriate ones. Firstly, I chose to implement the OpenVPN community to remotely reach our system. The main criterion for the selection was that it is an open-source, completely free project and will soon be part of the Linux kernel. I chose batman-adv as the routing protocol because it is designed for mesh networks, which will be the basis of my implementation.

Then, I created a test environment using virtual machines that could act as a live system. I tested how the system would react to possible node loss and investigated other edge cases as well. Next, I drew conclusions and prepared a system design for the live system that could solve the problem described in my introduction.

Taking into account the lessons learned from the test environment, I have designed the system. The second part of this thesis covers the implementation of the design what I did previously. I switched from a more bounded local KVM host to a cloud-based implementation. As I needed a more flexible and easily manageable environment.

To make the configuration of the nodes as simple as possible I developed a Python script. The script was based on an MVC design pattern to make the program more transparent and to better separate the layers. During the development of the program, I faced several difficulties. Since we wanted to control the configuration using an external manager machine, we had to configure each machine via SSH connection. Also, to make the configuration of the machines more user-friendly, I needed a more user-friendly way to manage the application (GUI). From this graphical interface, it was possible to add and delete nodes later on. It can display the current topologies using graphs, and all the machines in the network can be configured according to our specifications. Before running the program, it is necessary to predefine the tap interfaces or nodes to be created in the JSON file used for this purpose.

During the deployment phase, the topology was configured in a short time after calling the appropriate logic instructions. Then, I performed measurements to test this system. I selected a particular client node and examined the test cases from its "point of view". With each case, I tried to filter out as many failure modes as possible. First, I checked if the client could communicate with its neighbor and then with the client furthest away. I then deleted a VPN connection from the topology and later disconnected a node to see how the system reacts to the occasional network difficulties. After finalizing the script, I examined the potential of the software and architecture that I had created for development. OpenVPN's open-source code and its inclusion in the Linux kernel left huge potential. I would also like to simplify the JSON file needed to run the program.

To summarize my thesis, I have managed to design a complex system that can handle possible network difficulties. It can scale up and down following the requirements and high availability depending on the load. Moreover, I managed to make a rather complex architecture controllable by a simple user interface.

12. Irodalomjegyzék

[1] „Felértékelődött a távmunka a Covid19 árnyékában,” [Online]. Available: https://www.ksh.hu/docs/hun/xftp/idoszaki/koronavirus-tavmunka/index.html. [Hozzáférés dátuma: 29. 03. 2022].

[2] „Hálózati topológiák,” [Online]. Available: http://centroszet.hu/tananyag/szoftver/b_hlzati_topolgik.html. [Hozzáférés dátuma: 18. 04. 2022].

[3] „OverviewOfOpenvpn,” [Online]. Available: https://community.openvpn.net/openvpn/wiki/OverviewOfOpenvpn. [Hozzáférés dátuma: 19. 03. 2022].

[4] „B.A.T.M.A.N. advanced,” [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki/Wiki. [Hozzáférés dátuma: 10. 04. 2022].

[5] „B.A.T.M.A.N. protocol concept,” [Online]. Available: https://www.open-mesh.org/projects/open-mesh/wiki/BATMANConcept. [Hozzáférés dátuma: 18. 03. 2022].

[6] „Routing scenarios,” [Online]. Available: https://www.open-mesh.org/projects/open-mesh/wiki/Routing_scenarios. [Hozzáférés dátuma: 1. 03. 2022].

[7] „B.A.T.M.A.N. Advanced quick start guide,” [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki/Quick-start-guide. [Hozzáférés dátuma: 25. 02. 2022].

[8] „Network Wide Multi Link Optimization (technical documentation),” [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki/Network-wide-multi-link-optimization. [Hozzáférés dátuma: 25. 02. 2022].

[9] „Tutorial for OpenVPN TAP Bridge Mode,” [Online]. Available: https://www.aaflalo.me/2015/01/openvpn-tap-bridge-mode/. [Hozzáférés dátuma: 12. 04. 2022].

[10] „What is cloud computing?,” [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing/#cloud-deployment-types. [Hozzáférés dátuma: 13 10 2022].

[11] „What is ECS?,” [Online]. Available: https://support.huaweicloud.com/intl/en-us/productdesc-ecs/en-us_topic_0013771112.html. [Hozzáférés dátuma: 19. 09. 2022].

[12] „How the MVC architecture works in JavaScript,” [Online]. Available: https://blog.sessionstack.com/how-javascript-works-writing-modular-and-reusable-code-with-mvc-16c65cbd9f64. [Hozzáférés dátuma: 01. 10. 2022].

[13] „Interactive network visualizations,” [Online]. Available: https://pyvis.readthedocs.io/en/latest/introduction.html. [Hozzáférés dátuma: 24. 09. 2022].

[14] „Hálózati topológia: 6 megmagyarázott és összehasonlított hálózati topológia,” [Online]. Available: https://heritage-offshore.com/net-admin/halozati-topologia-6-megmagyarazott-es/. [Hozzáférés dátuma: 2. 03. 2022].

[15] R. S. Y. a. K. R. M. Praveen Likhar, „SECURING IEEE 802.11G WLAN USING OPENVPN AND ITS IMPACT ANALYSIS,” 2011.

[16] „Understanding how split tunneling works with OpenVPN Access Server,” [Online]. Available: https://openvpn.net/for/split-tunneling-with-access-server/. [Hozzáférés dátuma: 07 03 2022].

[17] „Understanding how split tunneling works with OpenVPN Access Server,” [Online]. Available: https://openvpn.net/for/split-tunneling-with-access-server/. [Hozzáférés dátuma: 07 03 2022].

[18] „What Is MVC,” [Online]. Available: https://blog.sessionstack.com/how-javascript-works-writing-modular-and-reusable-code-with-mvc-16c65cbd9f64. [Hozzáférés dátuma: 01. 10. 2022].

13. Ábrajegyzék

14. Mellékletek

A program futtatásához használt JSON fájl tartama:


```
{
 "nodes": [
  {
   "id": "1",
   "name": "SERVER 1",
   "BatmanIP": "10.10.10.10",
   "ConnIP": "                    ",
   "VPCIP": "192.168.1.124",
   "taps": [
    {
     "id": "1",
     "type": "s",
     "source": "1",
     "destination": "2",
     "port": "10011"
    },
    {
     "id": "2",
     "type": "s",
     "source": "1",
     "destination": "4",
     "port": "10014"
    },
    {
     "id": "3",
     "type": "k",
     "source": "1",
     "destination": "2",
     "port": "10022",
     "remote": "192.168.1.223"
    }
   ],
   "kulcs": "#\n# 2048 bit OpenVPN static key\n#\n-----BEGIN OpenVPN Static key
V1-----
\n99332be7e074b2e9837959d177cdb641\n6805d52b16b7c12fabc5b69ac8292ce7\
```

ndbccd7ff1a33c8978720332b5346c36b\naea138ac4ae0c9f0181cbda863700417\n7e044938645f96c0e7b718e0ed221cf3\n257a968dc9fde1652bf5ea01cd87c15b\n7f8853c51db969982155866124427ce6\nc24a6e899342b71fc0d14450c8a9c6b2\n66ea0f6ba5ae09d83a5d74be312923ba\n6b36b21c2533e2d64fa64047bbdff3f9\ncef3b944ba26ce47fb95044b2420d4a8\nb5a29087ba2027783d8e0c5fd1d4867c\nbe0aacd538e42cdfd4d573dbbb39c752\n738804cdee938362cae4589e7012fd68\n012f70803943af06b406e350479f44c7\n08a464bdb4ffe89b7e79969c53a7bf68\n-----END OpenVPN Static key V1-----\n"
  },
  {
   "id": "2",
   "name": "SERVER 2",
   "BatmanIP": "10.10.10.11",
   "ConnIP": "   ",
   "VPCIP": "192.168.1.223",
   "taps": [
    {
     "id": "1",
     "type": "s",
     "source": "2",
     "destination": "1",
     "port": "10022"
    },
    {
     "id": "2",
     "type": "s",
     "source": "2",
     "destination": "3",
     "port": "10023"
    },
    {
     "id": "3",
     "type": "k",
     "source": "2",
     "destination": "4",
     "port": "10042",
     "remote": "192.168.1.194"
    },

```
      {
        "id": "4",
        "type": "k",
        "source": "2",
        "destination": "1",
        "port": "10011",
        "remote": "192.168.1.124"
      }
     ],
     "kulcs": "#\n# 2048 bit OpenVPN static key\n#\n-----BEGIN OpenVPN Static key V1-----
\n99332be7e074b2e9837959d177cdb641\n6805d52b16b7c12fabc5b69ac8292ce7\ndbccd7ff1a33c8978720332b5346c36b\naea138ac4ae0c9f0181cbda863700417\n7e044938645f96c0e7b718e0ed221cf3\n257a968dc9fde1652bf5ea01cd87c15b\n7f8853c51db969982155866124427ce6\nc24a6e899342b71fc0d14450c8a9c6b2\n66ea0f6ba5ae09d83a5d74be312923ba\n6b36b21c2533e2d64fa64047bbdff3f9\ncef3b944ba26ce47fb95044b2420d4a8\nb5a29087ba2027783d8e0c5fd1d4867c\nbe0aacd538e42cdfd4d573dbbb39c752\n738804cdee938362cae4589e7012fd68\n012f70803943af06b406e350479f44c7\n08a464bdb4ffe89b7e79969c53a7bf68\n-----END OpenVPN Static key V1-----\n"
    },
    {
     "id": "3",
     "name": "KLIENS 3",
     "BatmanIP": "10.10.10.20",
     "ConnIP": "                    ",
     "VPCIP": "192.168.1.219",
     "taps": [
      {
        "id": "1",
        "type": "s",
        "source": "3",
        "destination": "5",
        "port": "10035"
      },
      {
        "id": "2",
        "type": "k",
```

```
        "source": "3",
        "destination": "2",
        "port": "10023",
        "remote": "192.168.1.223"
       }
     ],
     "kulcs": "#\n# 2048 bit OpenVPN static key\n#\n-----BEGIN OpenVPN Static key V1-----
\n99332be7e074b2e9837959d177cdb641\n6805d52b16b7c12fabc5b69ac8292ce7\ndbccd7ff1a33c8978720332b5346c36b\naea138ac4ae0c9f0181cbda863700417\n7e044938645f96c0e7b718e0ed221cf3\n257a968dc9fde1652bf5ea01cd87c15b\n7f8853c51db969982155866124427ce6\nc24a6e899342b71fc0d14450c8a9c6b2\n66ea0f6ba5ae09d83a5d74be312923ba\n6b36b21c2533e2d64fa64047bbdff3f9\ncef3b944ba26ce47fb95044b2420d4a8\nb5a29087ba2027783d8e0c5fd1d4867c\nbe0aacd538e42cdfd4d573dbbb39c752\n738804cdee938362cae4589e7012fd68\n012f70803943af06b406e350479f44c7\n08a464bdb4ffe89b7e79969c53a7bf68\n-----END OpenVPN Static key V1-----\n"
   },
   {
    "id": "4",
    "name": "KLIENS 1",
    "BatmanIP": "10.10.10.21",
    "ConnIP": "                    ",
    "VPCIP": "192.168.1.194",
    "taps": [
     {
      "id": "1",
      "type": "s",
      "source": "4",
      "destination": "2",
      "port": "10042"
     },
     {
      "id": "2",
      "type": "k",
      "source": "4",
      "destination": "1",
      "port": "10014",
```

```
        "remote": "192.168.1.124"
       },
       {
        "id": "3",
        "type": "k",
        "source": "4",
        "destination": "5",
        "port": "10056",
        "remote": "192.168.1.147"
       }
     ],
     "kulcs": "#\n# 2048 bit OpenVPN static key\n#\n-----BEGIN OpenVPN Static key V1-----
\n99332be7e074b2e9837959d177cdb641\n6805d52b16b7c12fabc5b69ac8292ce7\ndbccd7ff1a33c8978720332b5346c36b\naea138ac4ae0c9f0181cbda863700417\n7e044938645f96c0e7b718e0ed221cf3\n257a968dc9fde1652bf5ea01cd87c15b\n7f8853c51db969982155866124427ce6\nc24a6e899342b71fc0d14450c8a9c6b2\n66ea0f6ba5ae09d83a5d74be312923ba\n6b36b21c2533e2d64fa64047bbdff3f9\ncef3b944ba26ce47fb95044b2420d4a8\nb5a29087ba2027783d8e0c5fd1d4867c\nbe0aacd538e42cdfd4d573dbbb39c752\n738804cdee938362cae4589e7012fd68\n012f70803943af06b406e350479f44c7\n08a464bdb4ffe89b7e79969c53a7bf68\n-----END OpenVPN Static key V1-----\n"
    },
    {
     "id": "5",
     "name": "KLIENS 2",
     "BatmanIP": "10.10.10.22",
     "ConnIP": "                    ",
     "VPCIP": "192.168.1.147",
     "taps": [
      {
       "id": "1",
       "type": "k",
       "source": "5",
       "destination": "3",
       "port": "10035",
       "remote": "192.168.1.219"
      },
```

```
      {
        "id": "2",
        "type": "s",
        "source": "5",
        "destination": "4",
        "port": "10056"
      }
    ],
    "kulcs": "#\n# 2048 bit OpenVPN static key\n#\n-----BEGIN OpenVPN Static key
V1-----
\n99332be7e074b2e9837959d177cdb641\n6805d52b16b7c12fabc5b69ac8292ce7\
ndbccd7ff1a33c8978720332b5346c36b\naea138ac4ae0c9f0181cbda863700417\n7
e044938645f96c0e7b718e0ed221cf3\n257a968dc9fde1652bf5ea01cd87c15b\n7f88
53c51db969982155866124427ce6\nc24a6e899342b71fc0d14450c8a9c6b2\n66ea0f
6ba5ae09d83a5d74be312923ba\n6b36b21c2533e2d64fa64047bbdff3f9\ncef3b944
ba26ce47fb95044b2420d4a8\nb5a29087ba2027783d8e0c5fd1d4867c\nbe0aacd53
8e42cdfd4d573dbbb39c752\n738804cdee938362cae4589e7012fd68\n012f7080394
3af06b406e350479f44c7\n08a464bdb4ffe89b7e79969c53a7bf68\n-----END
OpenVPN Static key V1-----\n"
  }
 ],
 "Connections": [
  {
   "ConnIP": "                    ",
   "user": "root"
  },
  {
   "ConnIP": "                    ",
   "user": "root"
  },
  {
   "ConnIP": "                   ",
   "user": "root"
  },
  {
   "ConnIP": "                    ",
   "user": "root"
  },
```

```
  {
    "ConnIP": "                    ",
    "user": "root"
  }
 ]
}
```