



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Szepesi-Nagy István

HÁLÓZATI FESZÍTŐFÁK AUTOMATIZÁLT ÁTALAKÍTÁSA

KONZULENS

Dr. Zsóka Zoltán

BUDAPEST, 2021

Tartalomjegyzék

Összefoglaló	1
Abstract.....	2
1 Motiváció	3
1.1 Előzmények.....	3
1.2 STP használata a mai Enterprise vagy SOHO hálózatokban.....	5
1.3 Az automatizálás előnyei	7
2 STP protokollok áttekintése.....	9
2.1 STP.....	9
2.2 PVST+	10
2.3 RSTP	11
2.4 Rapid PVST+	12
2.5 MSTP	13
3 A hálózati feszítőfa kialakítása	14
3.1 A módszer célja.....	14
3.2 A módszer meghatározás	15
3.2.1 Algoritmus kidolgozása	15
3.2.1.1 Gráf felépítése a hálózati topológia alapján.....	17
3.2.1.2 Új optimális feszítőfa meghatározása	18
3.2.1.3 Hálózati paraméterek megváltoztatása	21
3.2.2 Esetleges variációk	24
3.2.2.1 Egyszerű feszítőfa algoritmus alkalmazása.	24
3.2.2.2 Különböző link-sebességek figyelembevétele.....	24
3.2.2.3 Legkevesebb változtatás figyelembevétele.....	24
3.3 A topológián való alkalmazás következménye	25
3.3.1 A root priorítás módosítása.....	25
3.3.2 A port költségek megváltoztatása.	27
4 Megvalósítás	28
4.1 Alkalmazott technológiák	28
4.1.1 NETCONF.....	28
4.1.2 NetworkX.....	31
4.1.2.1 Gráf létrehozása	32


4.1.2.2 Gráf kirajzolása	32
4.1.2.3 Különböző gráfalgoritmusok	33
4.2 A program működése	35
4.2.1 Main Graph	36
4.2.2 Device Info	37
4.2.3 Connector	38
4.2.4 Get Data	39
4.2.5 Shortest Path	40
4.2.6 Center Algorithm	41
4.2.7 Set Data	42
4.2.8 Szálkezelés	43
4.3 A program futtatási eredményei	44
4.3.1 A virtuális hálózati környezet bemutatása	44
4.3.2 Eredmények	46
4.3.2.1 Első eset	46
4.3.2.2 Második eset	51
5 Kiértékelés	54
5.1 Átlagos távolságok vizsgálata	54
5.2 Konvergencia idő mérése	57
5.3 Konklúzió	60
Irodalomjegyzék	62
Függelék	64

Hallgatói nyilatkozat

Alulírott **Szepesi-Nagy István**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 09.


.....
Szepesi-Nagy István

Összefoglaló

A második rétegbeli eszközök között a Spanning Tree Protocol biztosítja a hálózati feszítőfák kialakítását. Statikusan összekötött végpontok esetében az STP beállítások állandóak, azonban egy dinamikusan változó hálózatban szükséges lehet a kialakult feszítőfát átrendezni. Az átrendezés motivációi között lehet a teljesítmény növelése vagy a terhelésselosztás.

A szakdolgozatom feladata az önálló laboratórium során megoldott problémára épül. Előzményeként egy olyan rendszer automatizált konfigurálását valósítottam meg, ahol a hálózati felhasználók eszközei, dinamikusan átrendeződve csatlakoztak a hálózathoz. Ilyenkor a külön VLAN-ban lévő eszközök hozzáférését (switchport access) külön-külön engedélyezni kellett az egyes switch-eken, majd az előzőleg csatlakozott kapcsolókon törölni kellett a hozzáférési engedélyt biztonsági szempontból.

Egy komplex felépítésű L2 hálózatban elengedhetetlen az STP alkalmazása, azonban az előbb említett folytonos átalakítás mellett a kialakítása elmozdulhat az optimumtól. Erre a problémára találtam ki és implementáltam egy módszert Python programozási nyelven, ami a NETCONF protokoll felhasználásával valósul meg. A program automatizált rendszerben képes a hálózati topológia és az STP beállítások alapján új feszítőfákat felépíteni, az egyes VLAN-okra nézve. A megvalósítás során alkalmaztam egy Kruskal-algoritmuson alapuló maximális súlyú feszítőfa algoritmust, az új fa kialakítása érdekében pedig a switch-eken lévő port-költségeket és gyökér prioritásokat módosítottam.

A megvalósítás és tesztelés során egy virtuális hálózaton futó eszközök (router, switch, számítógép) segítségét használtam, valós berendezések hiánya miatt. Azonban a leírt megoldások és technológiák mindegyike a fizikai eszközök szintjén is megállja a helyét. A futtatási eredményeket pedig kettő szempont szerint elemeztem. Az átlagos távolságok vizsgálata az access switch-ek között szemlélteti a terhelési és a kialakítási eredményeket. A konvergencia időn alapuló vizsgálat pedig az új fa beállítási idejét tükrözi.

Abstract

Among the second layer devices, the Spanning Tree Protocol prevents the formation of bridge loops and the broadcast storm which results from them. For statically connected endpoints, the STP settings are constant, but it may be necessary to rearrange the resulting spanning tree in a dynamically changing network. The motivation for rearrangement can include the increasing of performance or load balancing.

The task of my thesis work is based on the problem solved during the Project Laboratory subject. As a precedent, I implemented an automated configuration of a system where the devices of network users were dynamically rearranged and connected to the network. In this case, the access of the devices in different VLANs had to be enabled separately on each switch, and then the access permission (switchport access) had to be revoked on the previously connected switches for security reasons.

In a complex L2 network, the use of STP is essential, but in addition to the continuous transformation, the design may differ from the optimum. I invented and implemented a method for this problem in the Python programming language, using the NETCONF protocol. In an automated system, the program is able to build new spanning trees based on the network topology and STP settings for each VLAN. During the implementation, I used a maximum weight spanning tree algorithm based on the Kruskal algorithm, and I modified the port costs and root priorities on the switches to create a new tree.

During the implementation and testing I used the help of devices running on a virtual network (router, switch, computer), due to the lack of real equipment, however, all the described solutions and technologies are also valid on physical devices. I analyzed the results from two perspectives. Examining the average distances between the access switches represents the load and design results. The convergence time test reflects the duration of the new tree to take place.

1 Motiváció

Több lépésen keresztül jutottam el arra a szintre, hogy a szakdolgozat feladatában kitűzött problémát képes legyek megoldani. Ehhez elengedhetetlen volt az elmúlt években szerzett önálló, vagy egyetemi úton szerzett tapasztalat.

Ebben a részben szeretném megemlíteni az eddigi munkámat, amelyet az Önálló laboratórium tantárgy keretében végeztem és bemutatni azt az automatizált hálózati feladatot, amely a szakdolgozatom előzménye és alapja. Emellett kitérek az STP (Spanning Tree Protocol) relevanciájára a mai nagyvállalati (enterprise) vagy SOHO (Small Office / Home Office) hálózatokban. Mindezek mellett az automatizálás előnyeit és alkalmazhatóságát is szeretném szemléltetni.

1.1 Előzmények

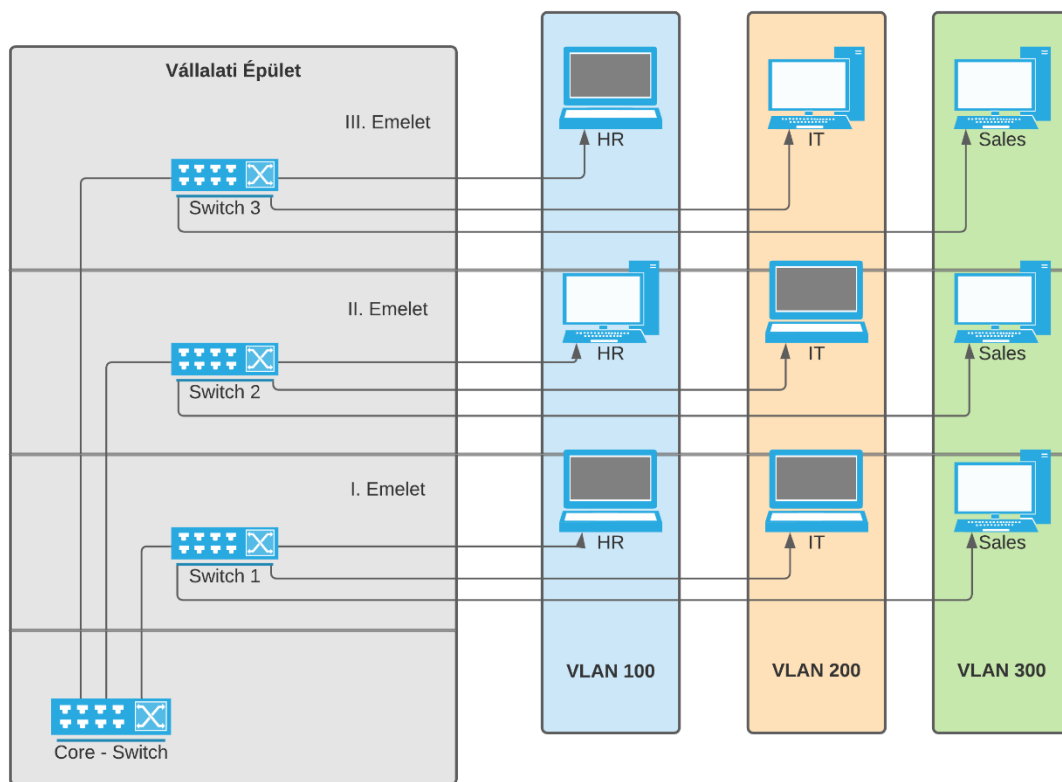
Az önálló laboratórium keretében egy olyan automatizált megoldást hoztam létre, amely a következő problémára volt alkalmazható.

Egy lokális hálózatban lévő L2 (Layer 2 [1]) switch-ekben az egyes interfészeket külön logikai hálózatokhoz, úgynevezett VLAN-okhoz (Virtual Local Area Network) tudjuk rendelni. Az ilyen szeparáció hálózattervezési és biztonsági szempontból is előnyös.

Egy egyszerű példa lehet a VLAN-ok használatára egy nagyvállalati hálózat, ahol az egyes osztályok (HR, IT vagy Sales) külön logikai hálózatokhoz vannak hozzárendelve, így mind a felhasználói végpontok, mind az osztályokhoz tartozó adatok szeparálva vannak egymástól. Ahogy a lenti ábrán is látszódik (ábra 1.1) nem szükséges azonos fizikai helyen legyenek a végfelhasználók, mert a switch-ek interfészeihez rendelt VLAN tökéletesen elegendő a megfelelő virtuális hálózathoz való kapcsolódáshoz. Persze megemlítendő, hogy a központi switch-hez tartó linkek úgynevezett trunk kapcsolatként (olyan kapcsolat, amely több VLAN forgalmát továbbítani képes) kell legyenek konfigurálva, illetve a VLAN-ok közötti kommunikációhoz mindenképpen szükséges egy L3 szintű - általában router vagy multilayer switch - berendezés is.

Akkor nincs probléma, ha a végfelhasználók fizikai helye mindig azonos, ilyenkor statikusan be lehet állítani a megfelelő VLAN-t az interfészen. Az önálló laboratóriumi feladat során azt a feltételezést tettem, hogy egy olyan vállalati épület létezik, ahol az ott

dolgozóknak nincsen fix helyük, vagyis bárhol csatlakozhatnak egy switch-en keresztül a hálózathoz. Ennek a kézi üzemeltetése sok feladatot igényel és csak lassan kivitelezhető.



ábra 1.1 - Nagyvállalati példa VLAN-ok alkalmazására.

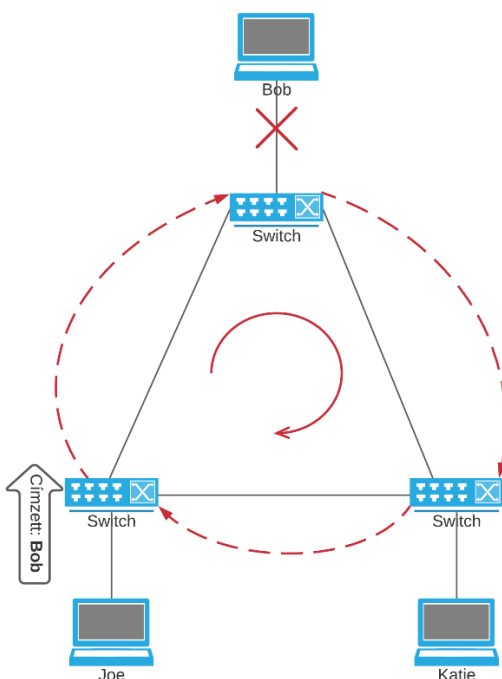
Az automatizálási megoldásom a fenti problémára egy adatbázis alapján dolgozó Python program volt, ami NETCONF (Network Configuration Protocol) használatával érte el az egyes switch-eket. Ha egy új switch-en vagy új interfészen jelent meg az eszköz, akkor az előző interfész lekapcsolt állapotba került, az új interfészhez hozzárendelte a program az adatbázisban szereplő megfelelő VLAN számot és frissítette az adatbázist. A rendszert egy virtualizált topológián is futtattam, ahol a link áthelyezése után a program indulásával az access portok (VLAN elérési portok) automatikusan átkonfigurálódtak.

A szakdolgozat feladatának kiválasztása során ez nyújtotta a fő motivációt, hiszen a folyamatos hálózati változás nem minden esetben optimális az STP (Spanning Tree Protocol) szempontjából. Projektemben azt vizsgálom, hogy a végfelhasználók folyamatos változása mellett milyen megfelelőbb feszítőfát lehet kialakítani az aktuálishoz képest.

1.2 STP használata a mai Enterprise vagy SOHO hálózatokban

A Spanning Tree Protocol azt biztosítja a második rétegbeli hálózatban, hogy a topológiában ne alakuljon ki kör, vagyis átviteli hurok. Ez azért fontos, mert L2 szinten nincs olyan mechanizmus, ami felügyelné egy keret (frame) maximális tartózkodási idejét a hálózatban. L3 szinten ilyen létezik, ezt hívják TTL (time-to-leave) értéknek, ha az értéke 0 lesz, akkor a csomag eldobásra kerül.

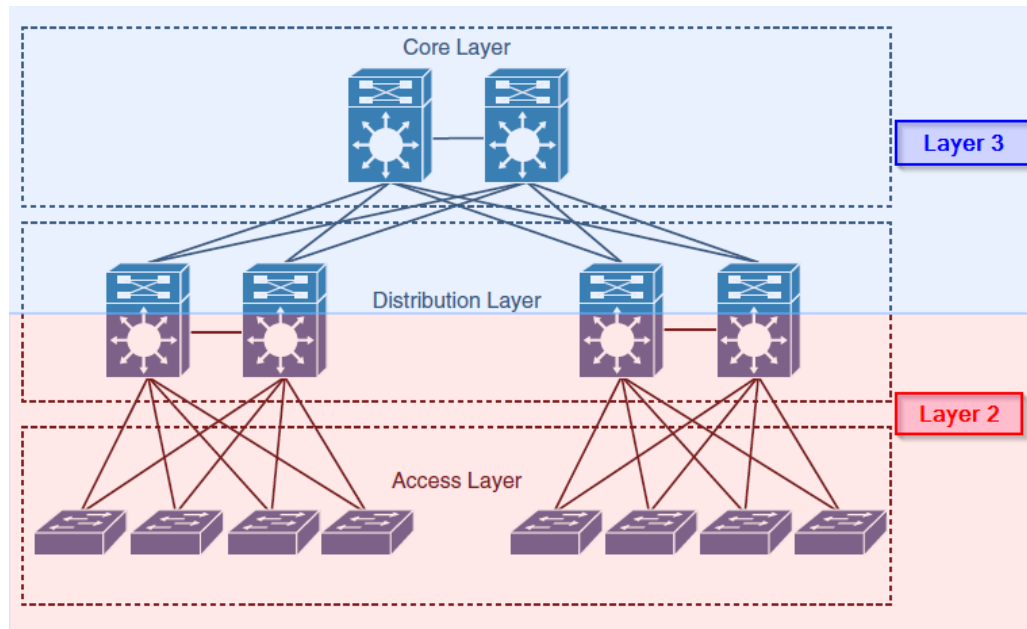
Ennek hiányában (ábra 1.2) viszont L2 szinten, ha egy olyan keret van jelen a hálózatban, aminek a végcélja hibás vagy nem létezik és hurok van a topológiában, akkor az üzenet örökké a hálózatban marad, mert egyik végpont se fogadhatja el. Ez a linkek túlterheléséhez vezethet, ami pedig keretek eldobását vonhatja maga után, vagyis a hálózat nem funkcionál tökéletesen.



ábra 1.2 - STP beállítás nélküli topológia | A Bob-nak címzett üzenet nem éri el a végfelhasználót, mert nincs csatlakoztatva a hálózathoz. A keret „örökké” a hálózatban marad.

A mai nagyvállalati vagy SOHO hálózatokban még mindig elterjedt az STP alkalmazása, ha a topológia megkívánja azt. Az adott hálózatokra mindig a megfelelő konfigurációk szükségesek (megfelelő STP protokoll használata, topológia kiválasztása, linkek sebessége), hogy a végfelhasználók eszközeikkel (PC, laptop) effektíven tudjanak a hálózat részesei lenni.

Az ilyen hálózatoknál, akkor jöhet szóba az STP alkalmazása, ha egy biztonságos topológiát építenek ki, ahol ki van zárva az SPOF (Single Point of Failure) és redundáns felépítés van jelen. Egy alkalmazható tervezési minta lehet a hierarchikus LAN tervezési minta (ábra 1.3), ahol a fenti elvárások megjelennek.



ábra 1.3 - Hierarchikus LAN tervezési minta. [2]

Egy vállalat vagy SOHO számára ideális választás lehet, mert skálázható és biztonságos felépítésű a redundancia okán. Ha azonban a switch-eken nem csak egy, hanem több VLAN van beállítva, illetve a vállalat nem használ multilayer switch-eket (L3 switch), akkor elkerülhetetlen az STP alkalmazása a hurkok kialakulása végett.

Egy másik megoldás lehet, ha az elérési réteg (access layer) aljáig terjed a Layer 3 szint, ilyenkor nincs szükség STP protokollra, hanem a switch-ek között route-olva haladnak a csomagok.

Tehát az STP használata nagyban függhet a cég vagyonától, eddigi eszközeitől és az igényeitől, mindazonáltal szerintem semmiképpen sem elhanyagolható az STP használata még az elkövetkező években.

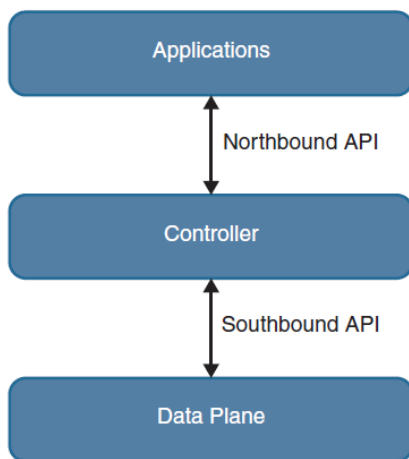
Nem nagyvállalati rendszereknél pedig, mint például az adatközpontokban (Data Center), eltérő tervezési mintával is dolgoznak (Spine-Leaf), ahol minden adat L3 szinten utazik, vagyis ezen rendszereknél szükségtelen az STP használata. [3]

1.3 Az automatizálás előnyei

Az automatizálás megjelenése előtt a hálózatzüzemeltetők feladatai közé tartoztak az eszközök megfelelő beállítása, az elvárt igényeknek megfelelően. A konfiguráció során az eszközökre valamilyen módon be kellett tudni lépni (pl.: konzol, Telnet vagy SSH kapcsolattal), majd a megfelelő parancsok kiadása után elmenteni a konfigurációs beállításokat. Ez a módszer még egy gyakorlott hálózatos szakember számára is igen lassú folyamat tudott lenni, ha egy bonyolult konfigurációt kellett létrehozni. Egy ember egyszerre csak egy eszközben tudott változtatásokat indukálni, illetve, ha valamilyen hibát ejtett, akkor az elmentett konfiguráció hibásan működött az eszközön.

Ezekre egy jó megoldást nyújthat az elmúlt években egyre jobban elterjedő SDN (Software Defined Networking) megoldások. Az alapvető különbség, hogy egy valós személy helyett egy központi kontroller konfigurálja az eszközöket. A kontroller egyszerre több eszközbe képes belépni és beállításokat módosítani, vagyis rendkívül jól skálázható, illetve a változtatási parancsok kiadása is szinte azonnali.

A kontrolleren kívül kettő fontos eleme egy SDN hálózatnak az úgynevezett Northbound - és Southbound API (Application Programming Interface) ábra 1.4. Az első a kontroller és a felhasználó által kezelt alkalmazás között húzódik. Ez lehetővé teszi, hogy a felhasználó által kívánt beállításokat küldje tovább a rendszer a kontroller felé. A Southbound API a kontroller és az egyes eszközök között húzódik. Ha sok eszköz van a rendszerben, akkor a kontroller az összes eszköz felé mind Southbound API-n keresztül kommunikál. A szakdolgozat feladata során a Southbound API-ként megvalósuló programot szeretném létrehozni. A hálózati eszközök menedzselésére és konfigurálására a következő protokollok és automatizációs szoftverek a legelterjedtebbek. [4]



ábra 1.4 - Az SDN rendszerek általános felépítése.

- Protokollok:
 - NETCONF - Network Configuration Protocol (SSHv2) [5]
 - RESTCONF - HTTP alapú NETCONF [6]
- Automatizációs szoftverek:
 - Puppet
 - Chef
 - SaltStack
 - Ansible

Ezen eszközökkel képesek vagyunk utasításokat és beállítási paramétereket küldeni az egyes eszközökre, ahol a változtatások azonnal életbe lépnek. Gyakran egy létező programnyelv segítségével (pl.: NETCONF esetében Python) történik a konfiguráció lekérdezése és megváltoztatása, de az egyes szoftverek rendelkezhetnek saját adatstruktúrákkal és beállításokkal, amelyekben a változtatni kívánt értékek vannak eltárolva. Ilyen adatstruktúra például az Ansible esetében az egyes playbook-ok, ahol különböző változtatásokra eltérő parancsokat vagyunk képesek eltárolni.

Ezekből következi, hogy manapság egy hálózatokkal foglalkozó szakembernek elengedhetetlen az alapvető programozási tudás, amit az SDN megjelenése előtt nem lehetett elmondani.

```

---
- name: change the hostname of SW-Core
  hosts: switches[0]
  gather_facts: no

  tasks:
    - name: hostname change
      cisco.ios.ios_system:
        hostname: SW-Core

- name: change the hostname of SW1
  hosts: switches[1]
  gather_facts: no

  tasks:
    - name: hostname change
      cisco.ios.ios_system:
        hostname: SW1

- name: change the hostname of SW2
  hosts: switches[2]
  gather_facts: no

  tasks:
    - name: hostname change
      cisco.ios.ios_system:
        hostname: SW2

```

ábra 1.5 - Példa Playbook felépítésére.

2 STP protokollok áttekintése

A Spanning Tree Protocol, ahogyan azt a 1.2-es fejezetben is említettem azt a célt szolgálja, hogy teljes körmentes részgráfot (feszítőfát) hozzon létre a linkek adattal való elömllesztésének elkerülése végett (broadcast storm). A switch-ek, ahol STP engedélyezve van, egymással BPDU (Bridge Protocol Data Unit) üzeneteket váltanak, annak érdekében, hogy tájékoztatni tudják egymás státuszát és adatait a fában. [2]

STP úgy építi fel a fát, hogy blokkolja azokat a portokat, amelyek kör kialakulásához vezetnének. Ehhez az STA (Spanning Tree Algorithm) algoritmust használja, ahol egy központi switch-hez (Root Bridge) képest számítja ki az utakat.

Az évek során több különböző STP implementáció jelent meg az éterben. Egyre gyorsabb és egyre több funkciót lefedni képes protokollokat fejlesztettek, legyen az Cisco által szabadalmaztatott technológia vagy az IEEE által létrehozott protokoll. Ebben a szekcióban a különböző STP protokollok jellemzőit szeretném összefoglalni, illetve kitérni arra, hogy melyik típust használtam a megvalósítás során és miért?

2.1 STP

Ez az eredeti 802.1D protokoll [7], amit az IEEE szervezet határozott meg 1990-ben. Ez a protokoll az alapja a következőkben taglalt változatoknak. Redundáns kapcsolatok létrehozásáért felelős protokoll, amit Common Spanning Tree (CST) néven is szoktak említeni. A mai hálózatokban alkalmazott STP-hez képest a CST csak egy feszítőfát tudott kiszámolni és fenntartani a hálózatban, illetve mindezt a használt VLAN-ok számától függetlenül tette. Más VLAN hálózatok optimális esetben eltérő feszítőfa topológiát igényelnek, így a 802.1D protokoll továbbfejlesztése elengedhetetlen volt a technológia fejlődés igényeihez idomulva.

A '90-es években használt hálózati eszközök memória - és processzorkapacitását jól használta az eredeti STP. A kevés számítás és adat fenntartásához elegendő volt az alacsonyabb energiaforrás. Azonban egy esetleges linkhiba esetén az új fa kialakításának ideje (convergence time) lassúnak bizonyult, nagyjából 30-50 másodpercet vett igénybe egy változás a hálózatban. Ilyenkor pedig az adatforgalom teljesen megállt és ez előnytelen a végfelhasználók szempontjából.

Az STP protokoll engedélyezése után a hálózat eszközei meghatározzák a Root Bridge (gyökér kapcsoló) szerepet, majd ez a switch üzeneteket továbbítja a többi switch felé, amit Bridge Protocol Data Unit-nak (BPDU) neveznek. A BPDU üzeneteket kapó switch-ek ezután az interfészein az alábbi port szerepeket tudják létrehozni:

- **Root port:** A nem gyökér switch-eken lévő port, ami a root felé mutat. Ezeken a portokon továbbítanak adatokat a switch-ek a gyökér felé
- **Designated port:** Az a kijelölt port egy switch-en, amelyik az adattovábbításban részt vesz, de nem a gyökér felé mutat. Minden port a Root Bridge-en designated port lesz
- **Blocked port:** Minden olyan port, ami nem root - vagy designated port. A blokkolás segítségével nem alakul ki kör a hálózatban

A szerepek mellett a portoknak különböző státuszokon kell végighaladniuk, mielőtt adatot továbbíthatnának. Öt STP port státusz létezik, amelyek a következők:

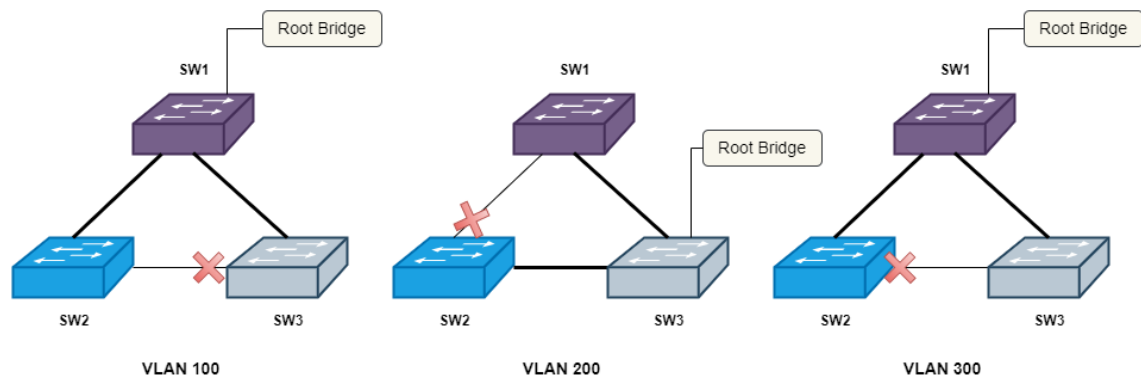
1. **Disabled:** Az adminisztrátor által kikapcsolt port státusza
2. **Blocking:** Amikor egy kapcsolat felépül először blokkoló státuszba kerül
3. **Listening:** A switch BPDU üzenetekre figyel, illetve továbbküld saját BPDU üzeneteket
4. **Learning:** A switch, ha egy felsőbb BPDU üzenetet kap, például a gyökértől, akkor továbbküldi azt, és ezentúl nem küld saját BPDU-kat
5. **Forwarding:** A port továbbítja az adatokat

2.2 PVST+

A CST továbbfejlesztésére alakult meg egy újabb protokoll, amit a Cisco Systems fejlesztett tovább, a Per VLAN Spanning Tree (PVST+). Alapvető funkciójában és tulajdonságaiban megegyezik a sima STP-vel, azonban ez a verzió már támogatja a VLAN-ok különválasztását az egyes fészítőfák kialakításakor.

Ez a funkció azért is előnyös egy hálózatban, mivel a logikai szeparáció fizikai térben is eltérhet különböző VLAN-okra, ahogy azt a 1.1 fejezetben is említettem. A PVST+ ezen felül a terhelést is segíti elosztani, illetve redundanciakihasználtság szempontjából is előnyös lehet.

A hálózatban lévő switch-eken eltérő VLAN-ok lehetnek engedélyezve vagy esetleg tiltva, és ilyenkor egy teljesen más fa lehet az optimális. A szakdolgozatomban én is ezt a felállást vizsgálom.



ábra 2.1 - Különböző VLAN-okra eltérő feszítőfa alakul ki a PVST+ protokoll segítségével.

2.3 RSTP

A Rapid Spanning Tree Protocol (RSTP), az eredeti STP-hez képest gyorsabb konvergencia időt biztosít, amit tipikusan 2 másodperc szokott lenni, azonban egy jól beállított hálózat ennél gyorsabb változtatási időt is képes produkálni. A 802.1w sztenderdet [8] az IEEE fogadta el, azonban a lassabb társával egyetemben szintén csak egy darab példányt tud fenntartani az összes VLAN-ra, ami nagyobb hálózatok esetén nem előnyös.

Az RSTP protokollnál már eltérő port szerepek és port státuszok vannak használatban. Ez a változtatás eredményezi azt, hogy a konvergencia idő radikálisan kisebb. Az STP-hez képest egyel több port szerep van életben, mivel a blokkolt állapot ketté vált úgynevezett *Alternate* és *Backup* szerepekre.

- **Root port:** A nem gyökér switch-eken lévő port, ami a root felé mutat. Ezeken a portokon továbbítanak adatokat a switch-ek a gyökér felé.
- **Designated port:** Az a kijelölt port egy switch-en, amelyik az adattovábbításban részt vesz, de nem a gyökér felé mutat. Minden port a Root Bridge-en designated port lesz.
- **Alternate port:** Megkapja a BPDU üzeneteket a többi eszköztől, azonban továbbra is blokkolt állapotban marad.
- **Backup port:** Csak saját magától kap BPDU üzeneteket és blocked szerepben marad.

A gyorsulás javítása érdekében a port státuszok is átalakításon estek át. A korábbi státuszokhoz képest, az RSTP protokollnál csak három fontos lépés létezik:

1. **Disabled:** Az adminisztrátor által kikapcsolt port státusza.
2. **Discarding:** Amikor egy kapcsolat felépül először discarding státuszba kerül. Ez az STP-nél a blocking státuszhoz hasonló állapot.
3. **Learning:** A switch, ha egy felsőbb BPDU üzenetet kap, például a gyökértől, akkor továbbküldi azt, és ezentúl nem küld saját BPDU-kat.
4. **Forwarding:** A port továbbítja az adatokat.

A lenti táblázat (táblázat 2.1) egy összefoglalás a megegyező és eltérő port-státuszokról. Látható, hogy a *blocking* és a *learning* státuszt felváltotta a *discarding*.

STP	RSTP
Blocking	Discarding
Listening	Discarding
Learning	Learning
Forwarding	Forwarding

táblázat 2.1 - STP és RSTP összehasonlítása.

2.4 Rapid PVST+

A következő STP protokoll a sorban a Rapid PVST+. Nevéből is adódik, hogy minden igényt igyekszik kielégíteni: gyors és minden egyes VLAN-ra képes példányt fenntartani. Ugyanakkor megemlítendő az is, hogy a processzor és memória igénye a legmagasabb az összes STP protokoll között, ez nem a legkorszerűbb switch-ekben vagy hatalmas számú VLAN-nal rendelkező hálózatban problémát jelenthet.

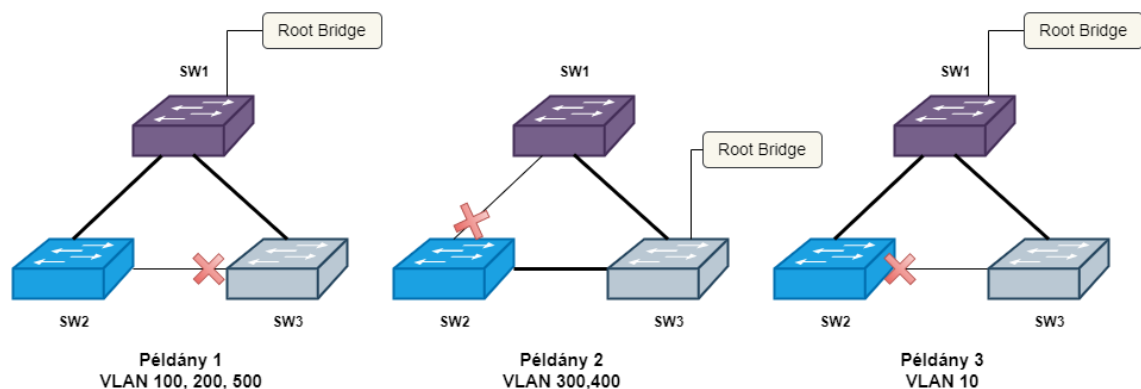
A szakdolgozati projektem megvalósításánál ezt a protokollt alkalmaztam és konfiguráltam fel a virtualizált eszközökre. A választást a feladat motivációja és a korábbi munkámhoz való kapcsolódás is indokolja. Mivel támogatni szükséges több, akár különböző elrendezésű VLAN-t is a hálózatban. Emellett a gyors konvergencia idő is támogatja a folyamatosan változó végfelhasználókat, illetve a folytonos újrakonfigurálást a rendszerben.

2.5 MSTP

Abban az esetben, ha egy vagy több VLAN-hoz tartozó feszítőfa alakja megegyező, felesleges lehet több példányt nyilván tartani és a switch erőforrásait feleslegesen használni. Erre a problémára hozták létre a Multiple Spanning Tree Protocol-t (MSTP). Egy adott példányhoz, több VLAN-t is hozzá tudunk rendelni, ha a fa alakja, a root switch és minden egyéb beállítás megegyezik (ábra 2.2).

A 802.1s [9] protokoll remek megoldás lehet, ha több száz, vagy ezer VLAN van a hálózaton, azonban a szakdolgozat projektjében létrehozott példamegoldásban nincs ennyi VLAN, illetve a különböző végpontok folyamatos változása is különböző alakú fákat eredményez, ezekhez pedig külön-külön példány kell. Ezért nem az MSTP mellett döntöttem.

Az erőforrás használata kevesebb a Rapid PVST+ -hoz képest, ami a kevesebb példány számából következik. Adatközpontokban vagy hatalmas kampuszhálózatokban érdemes lehet a használata, ha Spanning Tree protokollra van szükség.



ábra 2.2 - Példa MSTP példányok kialakítására.

A példában a Switch1 és Switch5 között lévő távolság kettő, míg Switch1 és Switch2 között egy. Ebben a szcenárióban az access switch-ek átlagos távolsága $\frac{4}{3}$. Cél tehát ezt a számot csökkenteni vagy minimalizálni az eddigi beállításokhoz képest.

Feltehető a kérdés, hogy miért is olyan fontos a folytonos változtatás, illetve mi értelme van a minimális távolságok felé való törekvésnek? Több szempont szóba jöhet a kérdés megválaszolásakor. Az első ilyen szempont a linkek terheltségének a kérdése: mivel törekszik a módszer a lehető legkevesebb inaktív linket bevonni az aktív kommunikációba (természetesen az egyes VLAN-okra nézve), ezért feleslegesen nincs adatforgalom olyan úton, amerre nem található végfelhasználó. Egy másik szempont lehet az esetleges adatszegmensek gyorsítása is, ha nem minden adat ömlesztve halad egy-egy linken, azonban ez lokális hálózatokban igen csekély és már-már elhanyagolható. A harmadik szempont pedig a már bemutatott automatizálás előnye (1.3. fejezet), hiszen a kézi hangolás lassú és fáradtságos munka lehet egy nagy hálózatot nézve.

3.2 A módszer meghatározás

Különböző módon lehetséges egy optimális feszítőfa-algoritmust megalkotni. A következőkben az általam kitalált rendszert szeretném részletesen bemutatni, illetve az okokat szemléltetni, melyeket a részfeladatok megoldásánál lefektettem.

Az én módszeremtől eltérő lehetséges algoritmusokat is szeretném áttekinteni, amelyek lehetnek jobb, vagy rosszabb megoldások is az általam választottnál.

3.2.1 Algoritmus kidolgozása

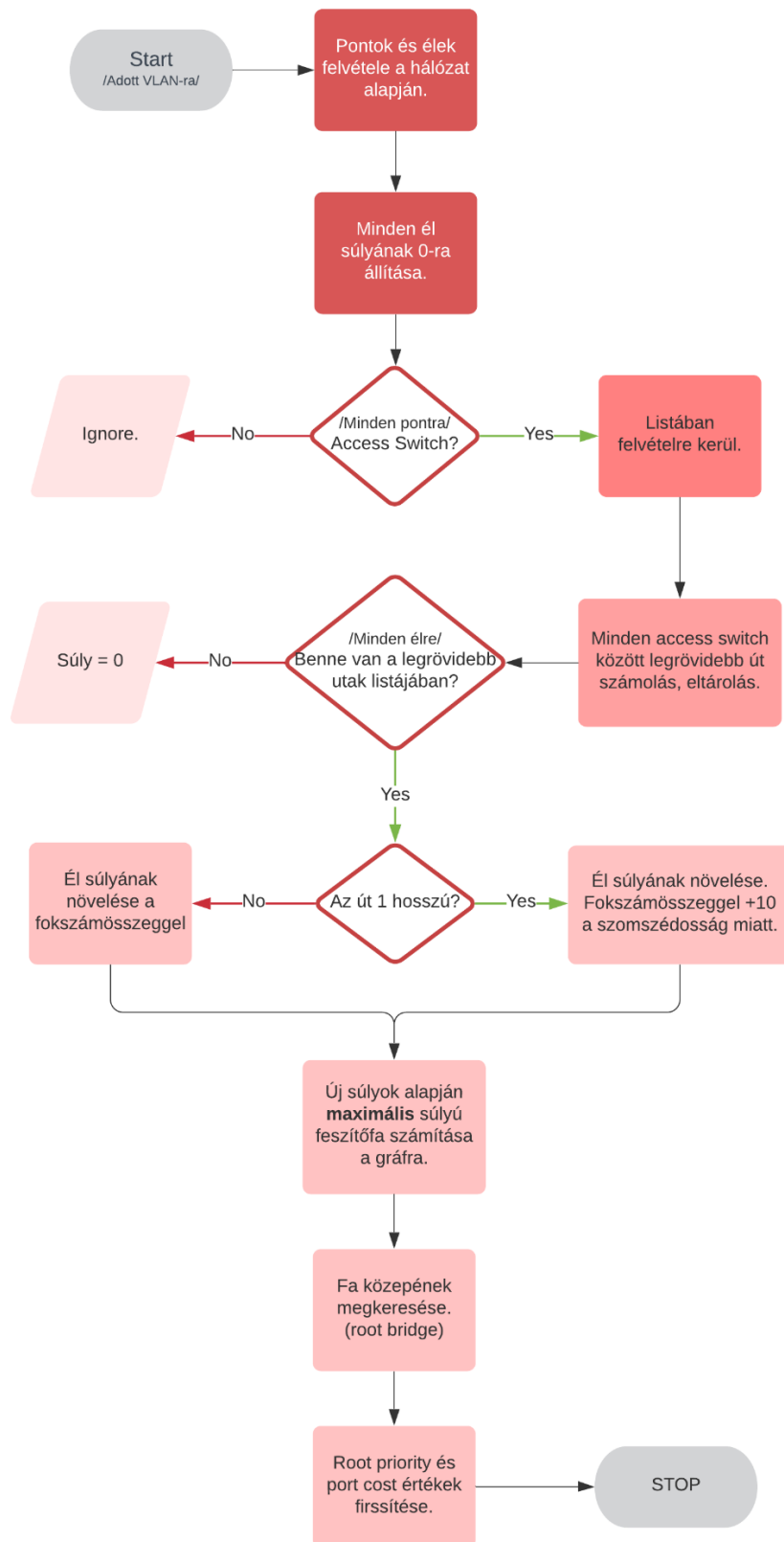
A feladat és a módszer célját mindig figyelembe véve jöttem elő egy programmal, ami optimalizálja az aktuális STP beállításokat. Az algoritmust strukturálisan három részre osztanám fel: gráf felépítése a hálózati topológia alapján, új optimális feszítőfa meghatározása, hálózati paraméterek megváltoztatása.

A program írása során több gráfalgoritmust alkalmaztam a NetworkX nevű Python könyvtárból (lásd NetworkX) az egyszerűség végett. Mint például a maximális súlyú feszítőfa algoritmus, vagy kettő pont közötti minimális távolság.

A módszerre készült algoritmus folyamatábráján (ábra 3.2) megfigyelhetőek az egyes lépések és a program egészének működése is. Az egyes folyamatokat az elkövetkezendő alfejezetekben részletezem.

Hálózati feszítőfa automatizált átalakítása

István Szepesi-Nagy | K45SFS | 2021.



ábra 3.2 - Az algoritmus folyamatábrája.

3.2.1.1 Gráf felépítése a hálózati topológia alapján

A hálózati eszközök elérési adatait, mint például IP-címek jelszavak és felhasználónevek, egy egyszerű struktúrában tárolom. Innen az összes switch adata könnyedén elérhető és ezen adatok segítségével létesíthető kapcsolat az eszközökkel. A kapcsolatok felépülése szálanként elkülönítve történik, így párhuzamosan tudja a program felvenni a gráfhoz tartozó pontokat a hálózat alapján.

Az élek felvétele kicsit érdekesebb történet. Ugyanúgy szálanként elkülönülve lépked be a program az eszközökbe, de itt nem csak önmagát kell visszaadja a switch, hanem az összes szomszédját is. A szomszédok nyilvántartása Cisco eszközökben a Cisco Discovery Protocol (CDP) segítségével valósul meg. Mivel csak Cisco által készített operációs rendszerű switch-csel dolgoztam, így ezt a megoldást alkalmaztam az élek felvételére. Természetesen az eszközökön engedélyezve kell legyen ez a funkció, hogy felfedezzék egymást a berendezések. Több gyártó termékét használó hálózatban a MAC-cím tábla lehet egy jó megoldás szomszédosság vizsgálatára. Az élekhez eltárolom az összeköttetéshez használt portok azonosító számát is, későbbi felhasználás végett.

A kialakult gráf tehát pont annyi ponttal és éllel rendelkezik, mint a valós (vagy virtuális) hálózat maga. A feszítőfa meghatározásánál ezt a gráfot veszem alapul, illetve ezen végzek kalkulációkat.

Még egy fontos lépés az előkészületeknél, hogy az összes olyan switch nyilvántartásba kerüljön, amely úgynevezett access switch (végponttal összekötött switch). Ennek felvételére kettő módszer is rendelkezésemre állt:

- A. Az önálló labor feladathoz hasonlóan egy külső egyszerű adatbázisban tárolhatom, hogy melyik eszköz, melyik switch, melyik interfészéhez kapcsolódik. Az eszközök a MAC-címük alapján vannak tárolva és a hálózati eszközök MAC-táblájából nyerek ki az információkat, az access switch lét meghatározásához.
- B. Engedélyezek az eszközökön egy PortFast Edge nevű funkciót, ami a végpontokkal kapcsolódó port-okon, nem engedélyezi az STP üzenetek továbbítását. A funkció engedélyezésével minden olyan port, ami közvetlen eléréssel (access port) van konfigurálva, kiesik az STP mezőjéből. Ilyenkor az adott VLAN-ra szűkített STP információk alapján egyértelműen azonosítható az a port, ami access módban van. (ábra 3.3)

```

SW1#sh spanning-tree vlan 100

VLAN0100
  Spanning tree enabled protocol rstp
  Root ID    Priority    4196
             Address     5254.0008.alaf
             This bridge is the root
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    4196  (priority 4096 sys-id-ext 100)
             Address     5254.0008.alaf
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  300 sec

Interface                Role Sts Cost      Prio.Nbr Type
-----
Gi0/0                    Desg FWD 4        128.1   P2p
Gi0/1                    Desg FWD 4        128.2   P2p
Gi1/3                    Desg FWD 4        128.8   P2p Edge
SW1#

```

ábra 3.3 - Access switch meghatározásához alkalmazott bejegyzés. | show spanning-tree vlan X eredménye alapján.

Az előző módszerek közül én a másodikat választottam, mivel ez a projekt az önálló laboratóriumi feladatnak csak egy továbbgondolása, így a két rendszert nem kötöttem össze. Ezek alapján, pedig a PortFast Edge bejegyzések alapján való azonosítás egyszerűbb megoldásnak bizonyult. Tehát az összes eszközön végighaladva megvizsgálom a közvetlen kapcsolatok létezését, és ha az adott kapcsoló access switch, akkor egy listába kerül felvételre.

3.2.1.2 Új optimális feszítőfa meghatározása

Az új feszítőfa meghatározásához először a felvételre került access switch-ek között húzódó legrövidebb utak felvétele szükséges. A listában lévő összes pont közötti legrövidebb utak száma mindig $\frac{n \cdot (n-1)}{2}$. Ezen utak mentésre kerülnek és újbóli vizsgálat során lesznek felhasználva.

A felvételre került G gráf éleinek súlyait pedig ezen legrövidebb utak alapján módosítja a program a következő táblázatban (táblázat 3.1) felsoroltak szerint, ahol $d_G(x)$ az x csúcshoz tartozó fokszámot jelöli. Minél nagyobb egy csúcs fokszáma, annál nagyobb súllyal számít a hozzá bekötődő él. A konstans tízzel való növelés pedig a szomszédosság jelentőséget növeli, hiszen kettő szomszédos access switch nagyobb eséllyel kell legyen összekötve. Az új súlyok ezután már a feszítőfa kialakításánál nyújtanak szerepet.

Eset	Súly
Az (x,y) él nem létezik a legrövidebb utakat tartalmazó listában.	0
Az (x,y) él létezik a listában, de az út nem 1 hosszú.	$d_G(x) + d_G(y)$
Az (x,y) él létezik a listában és az út 1 hosszú, vagyis a pontok szomszédosak.	$d_G(x)+d_G(y)+10$

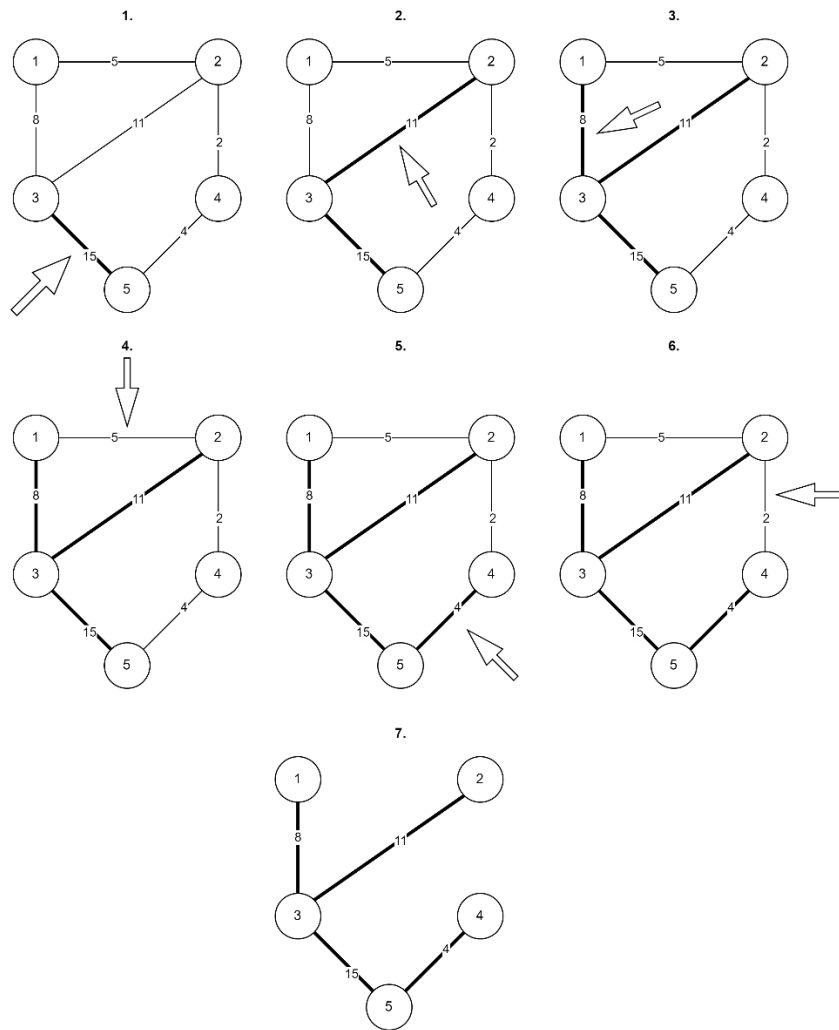
táblázat 3.1 - Élsúlyok megváltoztatásának esetei.

Ezek után a program a súlyok felhasználásával a G gráfra egy maximális súlyú feszítőfát számol ki, ami meghatározza a leghatékosabb körmentes teljes részgráfot a hálózatra nézve. A NetworkX könyvtárból felhasznált maximális súlyú feszítőfa algoritmus a Kruskal-algoritmust valósítja meg. [10]

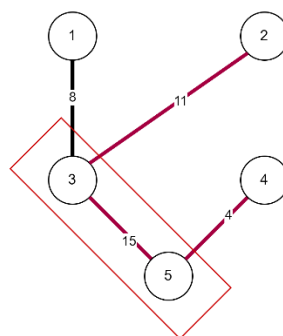
A Kruskal-algoritmus egy mohó algoritmus, abban a tekintetben, hogy mindig az első legjobb lehetőséget választja a soron következő élnek, és nem törődik a gráf többi részével az esetlegesen jobb megoldások miatt. A maximális súlyokat kereső Kruskal-algoritmus úgy működik (ábra 3.4), hogy sorban végighalad az éleken és a legnagyobb súlyú élet választja ki először. Ha már egy néhány nagy súlyú élet kiválasztott az algoritmus, azután kiválasztja a legnagyobb olyan élet, amely nem alkot kört a már eddig kiválasztottakkal. Ha ilyen él nincs, az algoritmus leáll, ha van folytatódik tovább a művelet. [11]

A kialakult feszítőfa meghatározása után a módszer alapján meg kell határozni a fa középső csúcsát. Erre azért van szükség, mivel a Root Bridge kapcsoló helye fontos lehet a valós hálózatban. Előnyös, ha egy központi helyen van elhelyezve, hiszen ő kezdeményezi az STP módosulások tényének üzeneteit. Ha a változások bekövetkezése után a gyökér a kialakult fa szélén lenne, az előnytelen lehet a hálózat működésének szempontjából.

A középső pont meghatározásához először a fában található leghosszabb utat kell megkeresni. Ez az összes csúcs között húzódó utak közül a leghosszabb, majd ezen leghosszabb útból kell kivenni a középső elemet (ábra 3.5). Értelemszerűen egy véges elemet tartalmazó listában a medián egy vagy kettő elem lehet. Ilyenkor azt a switch-et választja a program, amelyiknek a középső elemek közül nagyobb a fokszáma.



ábra 3.4 - Maximális súlyú feszítőfát megvalósító Kruskal-algoritmus folyamatábrája. A példa gráfon láthatóak az éleken elhelyezkedő súlyok, illetve az algoritmus által aktuálisan vizsgált él.



ábra 3.5 - A kialakult feszítőfában található középső elemek. Mivel a gráfban található leghosszabb út hossza páros, ezért kettő középső elem lesz a fa közepe. Ebben az esetben későbbi kiválasztás szükséges.

Egy fontos észrevétel, hogy a Root Bridge sajátosságai közé tartozik, hogy minden kimenő port, ami nincs PortFast Edge-ként konfigurálva, „designated” port kell legyen, ezért a középső elem fokszáma meg kell egyezzen az eredeti gráfban lévő

fokszámmal. Ennek érdekében a leghosszabb útból törlődnek, azok az csúcsok melyeknek fokszáma nem maximális. Ezután az így kialakult lista középső elemét kell megtalálni az előzőekben leírt mód alapján.

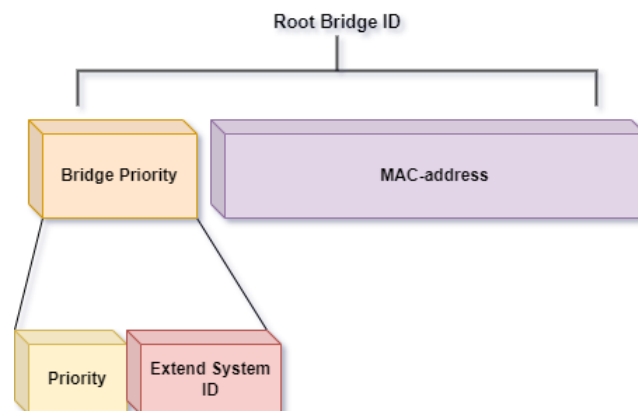
A középső csúcs megtalálásával és eltárolásával a program hozzá is kezdhet a változtatások érvényesítéséhez, amelyet a következő alfejezet ír le.

3.2.1.3 Hálózati paraméterek megváltoztatása

A kialakult feszítőfa és a megfelelő középső csúcs meghatározása után érvényesíteni kell a megváltozott paramétereket a hálózatban. Elsőként a Root Bridge szerepét frissíti a program, majd blokkolja azokat a linkeket, amelyek a fa szempontjából nincsenek használatban, az eddig blokkolás alatt levő linkeket pedig újra engedélyezi a topológiában.

A gyökér kapcsoló meghatározására az STP protokoll a Root Bridge ID azonosítót veszi figyelembe az egyes eszközökön (ábra 3.6). Az azonosító a gyökér switch prioritása, a rendszer prioritása (amit az adott VLAN száma határoz meg), illetve az eszköz MAC-címe alapján tevődik össze. A nyolc bájtos azonosító alapján a Root Bridge kiválasztása mindig a legalacsonyabb prioritás alapján történik, ha az összes eszköz prioritása megegyezik, akkor a legkisebb MAC-cím alapján történik a választás.

A prioritás csakis 0 és 61440 közötti, 4096-tal növelt érték lehet, ahol az alacsonyabb szám nagyobb prioritást eredményez. Az alapértelmezett érték 32768, amit a program azokon a switch-eken állít be, amelyeknek nincs Root Bridge szerepük. Az új gyökér elemnél pedig 4096-os érték kerül beállításra, így ez az eszköz lesz mindenképpen az STP közepe és irányítója.



ábra 3.6 - Root Bridge ID felépítése. A hálózat változtatásakor a *priority* mező értékét módosítja a program.

```

SW1#sh spanning-tree vlan 200
VLAN0200
Spanning tree enabled protocol rstp
Root ID    Priority    4296
           Address    5254.0005.6857
           Cost       8
           Port       1 (GigabitEthernet0/0)
           Hello Time 2 sec  Max Age 20 sec  Forward Delay 15 sec

Bridge ID  Priority    32968 (priority 32768 sys-id-ext 200)
           Address    5254.0008.a1af
           Hello Time 2 sec  Max Age 20 sec  Forward Delay 15 sec
           Aging Time 300 sec

Interface          Role Sts Cost      Prio.Nbr Type
-----
Gi0/0              Root FWD 4        128.1   P2p
Gi0/1              Desg FWD 100     128.2   P2p
SW1#

```

ábra 3.7 - Root Bridge ID értékek szemléltetése egy eszköz CLI kimenetén.

Root Bridge Priority = priority + sys-id-ext.

A következő szükséges változtatás a hálózatban az egyes összeköttetések engedélyezése vagy blokkolása. Erre a feladatra, az előző beállításhoz hasonlóan van egy szempontrendszer, ami alapján a port-ok prioritásait változtatni vagyunk képesek. A gyökér switch első lépésként úgynevezett BPDU (Bridge Protocol Data Unit) üzeneteket küld ki minden portján. A *root port* (ami a gyökér switch felé mutató interfész) kiválasztásánál a következő lépések játszanak szerepet:

1. A legkisebb útköltségű link előnyben részesített.
2. A legalacsonyabb rendszer prioritású switch, ahonnan a BPDU érkezett előnyösebb.
3. A legalacsonyabb MAC-címmel rendelkező, BPDU üzenetet küldő switch az előnyösebb.
4. Ha több link is hozzá van rendelve egy switch-hez, akkor az alacsonyabb port priority az előnyben részesített.
5. Ha több link is hozzá van rendelve egy switch-hez és a port prioritások megegyeznek, akkor az üzenetet küldő switch alacsonyabb számú port-ja az előnyben részesített.

A fenti szempontrendszer [12] alapján a legelső figyelembe vett paraméter a link útköltsége. Ezt a változót alkalmaztam a program megalkotásakor, mely szerint az alacsonyabb útköltséggel rendelkező linkek aktívan szerepet játszanak a feszítőfában,

míg a statikusan megemelt útköltségű linkek blokkolják az arra vezető forgalmat. Alapvetően az út költsége az egyes linkek sebességével állnak összefüggésben (táblázat 3.2), illetve kettő féle költségformátum létezik, amelyekből én a *Short-Mode* alakot használtam. Ezen felül a módszer jelenleg nem veszi figyelembe az eltérő sebességű linkeket. Feltételeztem, hogy a hálózat összes összeköttetése azonos sebességű (pl.: 1 Gbps)

Link Speed	Short-Mode STP Cost	Long-Mode STP Cost
10 Mbps	100	2,000,000
100 Mbps	19	200,000
1 Gbps	4	20,000
10 Gbps	2	2,000
20 Gbps	1	1,000
100 Gbps	1	200
1 Tbps	1	20
10 Tbps	1	2

táblázat 3.2 - Link sebességek és útköltségük rövid - illetve hosszú formátumban. [13]

Az útköltségek minden esetben a gyökértől számíthatók és összeadódnak, ezért ha az egyik port értékét egy elönytelenül nagy számra állítja a program, akkor az adott link kiesik a választási lehetőségek közül.

A kizárni kívánt útvonalak költségeit 100-as értékre, míg a rendes módban üzemelő linkekét 4-es értékre állítja a program. Ebből az következik, hogy a kívánt fa alakja megegyezik a program által kiszámolt fa alakjával, illetve az előzőekben módosított értékek is újra felül lesznek írva.

Ezzel minden szükséges beállítás módosításra került egy adott VLAN-ra nézve, az optimálisabb feszítőfa szempontjából. Természetesen ugyanezen lépések lezajlanak a többi létező VLAN esetében is.

3.2.2 Esetleges variációk

Az általam kialakított algoritmusok sorozata egy lehetséges változat a sok közül. Más megközelítésből is épülhet ki új feszítőfa azonban kérdéses, hogy mennyire lesz optimális az eddigi beállításokhoz képest. Egyéb lehetséges megvalósítások a következők lehetnek.

3.2.2.1 Egyszerű feszítőfa algoritmus alkalmazása.

A switch-ek alapján felvett csúcsok és élek között egyszerű minimális feszítőfa algoritmust alkalmazunk. Mivel az egyszerű feszítőfa algoritmus a felvétel sorrendjében halad végig a csúcsokon, ezért érdemes lehet egy egyszerű rendezést elvégezni a MAC-címek alapján, hogy az STP protokoll alapelveit is figyelembe vegyünk, mely szerint az alacsonyabb MAC-címmel rendelkező eszközök előnyösebbek (3.2.1.3). Az így kialakult fa kerülhet érvényesítésre ezek után.

Ezen variációban lényegtelen az optimalizálás vizsgálata, hiszen csak egy algoritmust (pl.: Kruskal - vagy Prim-algoritmus) alkalmazunk, de előfordulhat olyan eset, amikor éppen az általam kialakított módszerrel megegyező feszítőfa kerül felépítésre.

3.2.2.2 Különböző link-sebességek figyelembevétele.

Létezhet egy olyan variáció is, ahol az program figyelembe veszi a különböző sebességű linkeket is. Ilyenkor egy olyan út kerülhet kialakításra, amelynek az összsebessége maximális.

Ha különböző típusú kábelekkkel van egy hálózat megtervezve és kiépítve, akkor előnyös lehet ez a megközelítés és az általam javasolt módszerrel ötvözve még optimálisabb kialakítást lehet vele elérni. Viszont, ha az összes port sebessége azonos (pl.: GigabitEthernet), akkor ez a variáció is ugyanazt a kialakítást eredményezi.

3.2.2.3 Legkevesebb változtatás figyelembevétele.

Az ideális variáció egy olyan megoldás lehetne, ahol az aktuálisan életben lévő elrendezéshez képest, a legkevesebb olyan változtatást kellene elvégezni ahhoz, hogy az új felállítás során is a lehető legjobb legyen az STP. Ez egy magasabb szintű megvalósítás, hiszen folyamatosan monitorozni szükséges a hálózatot. Szerintem a három variáció közül ez a verzió eredményezné a legjobban a folytonos változás lekövetését, azonban

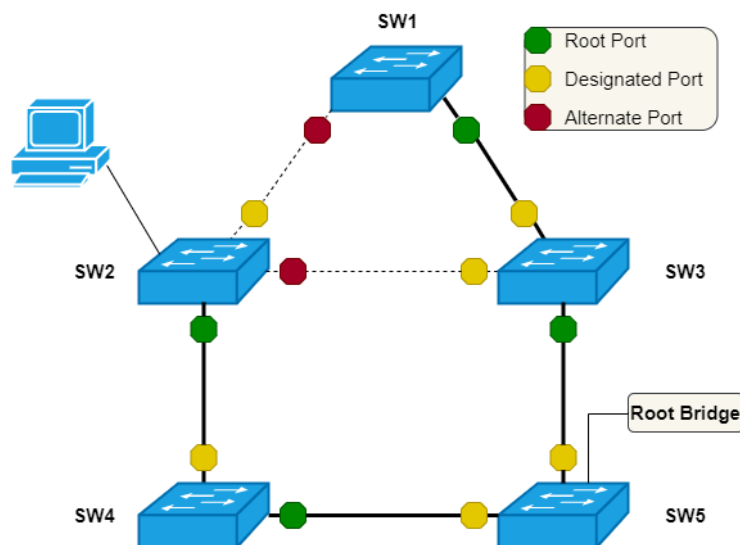
felmerülhet a kérdés ebben az esetben, hogy megéri-e kidolgozni egy ennyire összetett programot az adott problémára?

3.3 A topológián való alkalmazás következménye

A módszer befolyásolja az egyes beállítási paramétereket az eszközökön és a változások eredményeként kialakul az új feszítőfa. Olykor a változások befolyásolási eredménye nem triviális, ezért szeretném itt bemutatni, hogy mit jelent, illetve mit kell figyelni egy változás során. A root priority és a port cost attribútumok értékeit módosítom egy példán szemléltetve.

3.3.1 A root prioritás módosítása

Az egyszerűség és átláthatóság végett egy egyszerű példa topológián szeretném bemutatni ahogy a Root Bridge szerep megváltozik a hálózaton. Ehhez a következő felállást (ábra 3.8) kellene tanulmányozni, ahol jelölve vannak a portok szerepei, a csatlakozó eszközök és hogy melyik switch az aktuális root.



ábra 3.8 - A hálózati topológia alapfelállása az egyes port-ok szerepeivel és a root bridge eszközzel feltűntetve.

A kiinduló állapotban SW5 switch a root, amit a CLI (Command Line Interface) felületre belépve is tudunk ellenőrizni (ábra 3.9). Az eszközön a root prioritásnak 4096 van beállítva és mivel az aktuális VLAN száma 100, így az eszköz prioritás 4196 értékű lesz. A hálózaton ezen eszközé a legalacsonyabb, ezért került SW5-re a választás.

```

SW5#sh spanning-tree vlan 100

VLAN0100
  Spanning tree enabled protocol rstp
  Root ID    Priority    4196
             Address    5254.0010.15ed
             This bridge is the root
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    4196   (priority 4096 sys-id-ext 100)
             Address    5254.0010.15ed
             Hello Time  2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  300 sec

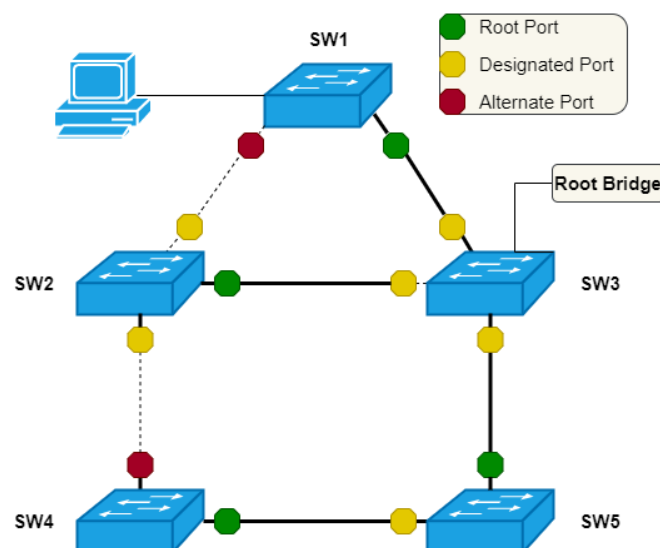
Interface                Role Sts Cost      Prio.Nbr Type
-----
Gi0/1                    Desg FWD 4        128.2 P2p
Gi0/3                    Desg FWD 4        128.4 P2p
SW5#

```

ábra 3.9 - SW5 eszköz CLI felülete. A 100-as VLAN-hoz tartozó STP információkkal feltüntetve.

Az eszköz prioritása 4196, amivel a hálózaton megkapta a Root Bridge szerepet.

Ha a személyi számítógép egy új eszközhöz csatlakoztatjuk és a topológia változik, akkor a root szerepe is változhat a hálózatban, az adott VLAN-t tekintve. A következő ábrán szemléltetem az új hálózati felállást (ÁBRA).



ábra 3.10 - Hálózati STP kialakítás a változtatások után. Az új Root Bridge switch SW3 lett, miután a PC csatlakoztatási helye megváltozott.

Az eredményeket ismét a CLI felületen tudjuk ellenőrizni (ábra 3.11). Az eddigi root (SW5) prioritás megváltozott 32868-ra, míg az új gyökér prioritása 4196 lett. A változás utáni különbség a lenti összefoglaláson is megfigyelhető (ábra 3.11). Fontos, hogy az eszközök nyilván tudják tartani, hogy melyik eszköz a Root Bridge, mivel tőle kapják a BPDU üzeneteket. Ha pedig maga a root változik, akkor először fel kell fedezzék a körülöttük lévő eszközök identitását, majd folytatódhat a rendes működés.

SW3#sh spanning-tree vlan 100

VLAN0100

Spanning tree enabled protocol rstp

Root ID

Priority

4196

Address

5254.000f.73ac

This bridge is the root

Hello Time

2 sec

Max Age 20 sec

Forward Delay 15 sec

Bridge ID

Priority

4196

(Priority 4096 sys-id-ext 100)

Address

5254.000f.73ac

Hello Time

2 sec

Max Age 20 sec

Forward Delay 15 sec

Aging Time

300 sec

Interface	Role	Sts	Cost	Prio.Nbr	Type
Gi0/0	Desg	FWD	4	128.1	P2p
Gi0/1	Desg	FWD	4	128.2	P2p
Gi0/3	Desg	FWD	4	128.4	P2p

SW3#

SW5#sh spanning-tree vlan 100

VLAN0100

Spanning tree enabled protocol rstp

Root ID

Priority

4196

Address

5254.000f.73ac

Cost

4

Port

4 (GigabitEthernet0/3)

Hello Time

2 sec

Max Age 20 sec

Forward Delay 15 sec

Bridge ID

Priority

32868

(Priority 32768 sys-id-ext 100)

Address

5254.0010.15ed

Hello Time

2 sec

Max Age 20 sec

Forward Delay 15 sec

Aging Time

300 sec

Interface	Role	Sts	Cost	Prio.Nbr	Type
Gi0/1	Desg	FWD	4	128.2	P2p
Gi0/3	Root	FWD	4	128.4	P2p

SW5#

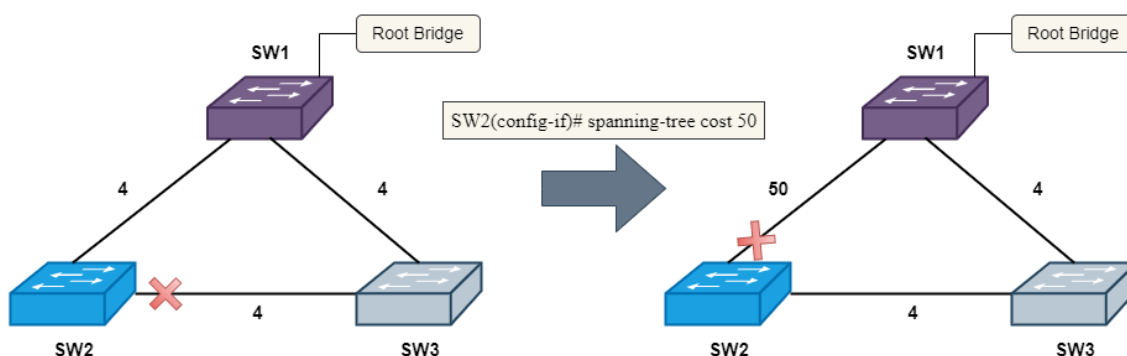
ábra 3.11 - Összehasonlító kimutatás SW3 és SW5 eszközök között. Az eddigi Root Bridge szerep megváltozott, az új gyökér SW3 lett, aminek a prioritása 4196.

3.3.2 A port költségek megváltoztatása.

A port költségek megváltoztatásával a program formálni képes a fa alakját. Ehhez tudni kell a switch megfelelő interfészének a számát, ennek tudatában pedig át lehet állítani a költséget az alábbi parancs segítségével.

SW(config-if)# spanning-tree cost X

A költség növelésével blokkolhatjuk egyes utakat, hogy az STP-n más felállás alakuljon ki. Például, ha szeretnénk megcserélni egy adott switch-hez tartozó útvonalat az egyik port-ról a másikra, akkor a blokkolni kívánt port költségét felemelhetjük egy igen magas számra, ami nem előnyös útvonalválasztást jelent (ábra 3.12). Természetesen, ha a switch egy levél a fában és csak egy link-en keresztül kapcsolódik a hálózathoz, akkor a költségnek nincs jelentősége.



ábra 3.12 - Port költség növelésének eredménye. Az eddigi közvetlen útvonal helyet SW2 inkább SW3-mon keresztül továbbítja a szegmenseket, mivel arra kisebb az összköltség ($8 \ll 50$).

4 Megvalósítás

A módszer meghatározása és bemutatása után ebben a részben szeretném részletesebben felvázolni az elkészült program egyes részeit, illetve a megoldáshoz felhasznált technológiákat. Majd szeretném lefuttatni a programot, hogy szemléltessem a működését.

4.1 Alkalmazott technológiák

Különböző technológiákat felhasználva írtam meg a működő programot. Többek között szükség volt a Cisco típusú switch-ek CLI-ben való konfigurálásának ismeretére. Szerencsére ebben a részben már volt gyakorlatom, így az egyes eszközök alapbeállításait könnyedén el tudtam végezni.

Az eszközök eléréséhez és központosított konfigurációjához a NETCONF protokollt alkalmaztam. Ezen protokoll megvalósítása egy egyszerűen alkalmazható Python könyvtárként valósult meg, az adatok feldolgozása pedig a NetworkX nevű könyvtár funkcióival történt.

4.1.1 NETCONF

A NETCONF protokoll egy IETF által definiált szabvány [14], ami különböző hálózati eszközök konfigurációinak telepítéséért, változtatásáért és törléséért felelős. A protokoll előzménye az SNMP (Simple Network Management Protocol) volt, ami több szintű hierarchiába rendezett utasítási módokból épült fel. Az SNMP használata nem volt elterjedt, mivel nem szövegalapú volt a felépítése, legtöbbször csak monitorozási céllal alkalmazták. Ezért hálózatos szakemberek továbbra is a CLI elérésen keresztül konfigurálták a berendezéseket. Realizálták ezt az IETF közösségben és 2002-ben elkezdték az SNMP protokoll újragondolását. [15]

A kialakított NETCONF protokoll már emberi nyelven értelmezhető segítséget nyújt a hálózati eszközök menedzselésében. A NETCONF négy rétegre van felbontva, amelyek lehetővé teszik a biztonságos és stabil kommunikációt az eszköz és a controller között. A rétegeket a táblázat 4.1 szemlélteti, illetve a következőkben kerülnek megemlítésre:

1. **Secure Transport Layer:** Ezen réteg segítségével egy biztonságos kapcsolat jön létre a kliens eszköz és az irányító szerver között.
2. **Messages Layer:** Üzeneteket tartalmazó és titkosító réteg, amely RPC (Remote Procedure Call) üzenetekbe beágyazva valósul meg. Az üzenet lehet RPC-hívás, RPC-válasz vagy valamilyen eseményről kapott értesítés. Az üzenet XML (Extensible Markup Language) formátumban van az RPC-be csomagolva.
3. **Operations Layer:** Ez a réteg különböző utasításokat tartalmaz, amelyeket a táblázat 4.1 is tartalmaz. Ezen utasítások törzse az előző rétegbeli üzenetek megfelelő XML formátumban.
4. **Content Layer:** Az eszközök adatait tartalmazó réteg. A YANG adatmodell használatos ezen rétegben való alkalmazásra.

	Layer	Example	
4.	Content	Configuration Data	Notification Data
3.	Operations	<edit-config>, <get-config>, <copy-config>, <delete-config>, <close-session>, ...	
2.	Messages	<rpc>, <rpc-reply>	<notification>
1.	Secure Transport	SSH, TLS, BEEP/TLS, SOAP/HTTP/TLS, ...	

táblázat 4.1 - NETCONF protokoll rétegei példákkal. [16]

A NETCONF kapcsolat egyszerű és programozott megoldására az *ncclient* nevű Python könyvtár [17] metódusait használtam. Az *ncclient*-ben minden olyan tulajdonságot biztosít, amit az előzőekben leírt rétegek tartalmaznak.

Először is a Cisco eszközök és az *ncclient* kapcsolódási függvénye is elvárja, hogy SSHv2 típusú biztonságos kapcsolat létesüljön a program és a switch-ek között. Az SSH 2-es verzióját minden egyes eszközön engedélyezni kell, hogy NETCONF segítségével programozhatók legyenek a berendezések. Az alap SSH-hoz képest a 2-es verzió biztonságosabb és hatékonyabb kapcsolatot biztosít.

Az XML leíró nyelvben nincsenek előre definiált címkék (A HTML-lel ellentétben), hanem saját címkéket lehet létrehozni a program igényeihez mérten. Strukturált formába lehet rendezni az adatokat, illetve ebben a rendezett formában

továbbküldeni vagy megosztani. A NETCONF esetében az RPC üzenetek vannak becsomagolva XML formátumba, amit az eszközök értelmezni tudnak.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:writeable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:startup:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:url:1.0</capability>
    <capability>urn:cisco:params:netconf:capability:pi-data-model:1.0</capability>
    <capability>urn:cisco:params:netconf:capability:notification:1.0</capability>
  </capabilities>
  <session-id>321472588</session-id>
</hello>]]>]]>
```

ábra 4.1 - Példa XML formátum egy RPC üzeneten szemléltetve.

A YANG adatmodell egy strukturált leíró nyelv, amelyet a NETCONF konfigurációk segítésére hoztak létre [18]. Az IETF-en belül a NETMOD felel a YANG nyelvért, amelyet több hálózati berendezést gyártó cég is használ az eszközeik menedzselésére. A konfigurációs adatok elkülönítve, minden egyes rész külön részt kap egy YANG adatstruktúrában. Például az eszköz interfészeinek beállítása és azok paraméterei elkülönítve vannak a routing beállításoktól.

```
module acme-system {
  namespace "http://acme.example.com/system";
  prefix "acme";

  organization "ACME Inc.";
  contact "joe@acme.example.com";
  description
    "The module for entities implementing the ACME system.";

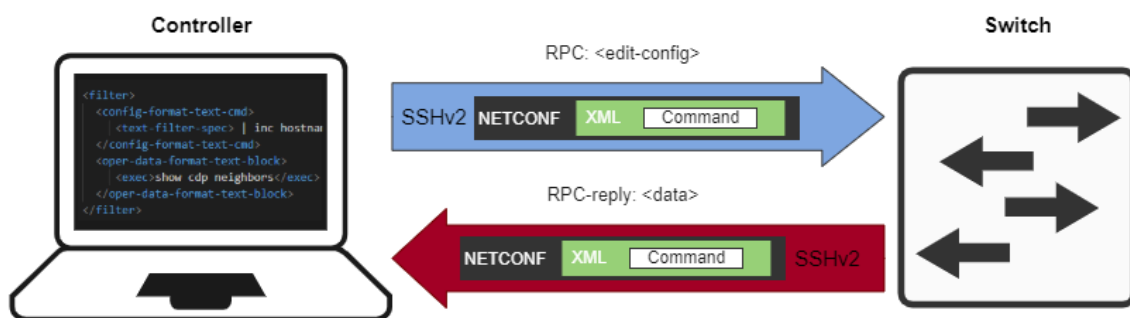
  revision 2007-11-05 {
    description "Initial revision.";
  }

  container system {
    leaf host-name {
      type string;
      description "Hostname for this system";
    }
    leaf-list domain-search {
      type string;
      description "List of domain names to search";
    }
    list interface {
      key "name";
      description "List of interfaces in the system";
      leaf name {
        type string;
      }
      leaf type {
        type string;
      }
      leaf mtu {
        type int32;
      }
    }
  }
}
```

ábra 4.2 - Példa YANG struktúrára. Az egyes utasítások konténerekben vannak tárolva, az elemek pedig levelekben. Ez a felépítés elősegíti a gyors keresést az adatmodellben. [19]

A NETCONF ezen struktúra felhasználásával képes változtatásokat gyorsan és effektíven végrehajtani az eszközökön. A támogatott adatmodellek listája minden esetben lekérhető az eszköztől. Egyes eszközök több adatmodellt támogatnak és vannak olyanok, amelyek kevesebbet. A megvalósítás során használt IOS rendszerű switch-ek kevés képességgel rendelkeztek ezért a YANG adatmodellek helyett az Operations rétegbeli utasításokkal tudtam megoldani a konfigurációk megváltoztatását. Ez azt jelenti, hogy valós parancsokat küldi el a NETCONF rendszere az eszközökhöz (tipikusan <edit-config> címkével ellátva) és így lépnek életbe a változtatások.

A NETCONF kapcsolat felépülésének és a konfiguráció módosításának is megvan a maga menete. Többszörös becsomagolás után a kontroller kiküldi az eszközöknek az utasításokat, majd a switch-ek vagy végrehajtják a változtatásokat (<edit-config>) vagy visszaküldik a kérés eredményét (<get-config>) további feldolgozásra. A kommunikáció folyamatára jó szemléltető példa az alábbi ábra (ábra 4.3).



ábra 4.3 - NETCONF kommunikáció folyamata.

4.1.2 NetworkX

A NetworkX (Network Analysis in Python) egy programcsomag, amely különböző bonyolultságú hálózatokkal képes dolgozni. Ezen hálózatokat képes létrehozni, manipulálni és vizsgálni a változásokat. [20]

A szakdolgozat feladata során ezt a könyvtárat alkalmaztam a hálózat felvételére, majd a kialakult gráfon való számolásokhoz és módosításokhoz is felhasználtam. Több, gyakran használatos gráfalgoritmust magában foglal a modul, ami nagyban megkönnyíti a hálózatokkal való dolgozást.

A NetworkX arra is képes, hogy a kialakított gráfokat vagy részgráfokat kirajzolja a Matplotlib könyvtár segítségével. Az eredmények ellenőrzése és a szemléltetés végett én is alkalmaztam a megjelenítési funkciókat.

4.1.2.1 Gráf létrehozása

A gráfok létrehozása a `Graph()` függvény hívásával indukálható. Ezután pedig a csúcsok és az élek felvételére biztosít lehetőséget a `NetworkX`. Az egyszerű statikus attribútumok megadásán kívül, lehetőség van nevezetes gráfokat létrehozni a könyvtár segítségével (pl.: K_5 teljes gráf vagy $K_{3,3}$ páros gráf).

```
>>> P = networkx.Graph()

>>> names = ["A", "B", "C", "D", "E"]
>>> P.add_nodes_from(names)
>>> P.add_edges_from([
    ("A", "B"),
    ("B", "E"),
    ("C", "A")
])
>>> print(P)
Graph with 5 nodes and 3 edges

>>> K = networkx.complete_graph(5)
>>> print(K)
Graph with 5 nodes and 10 edges
```

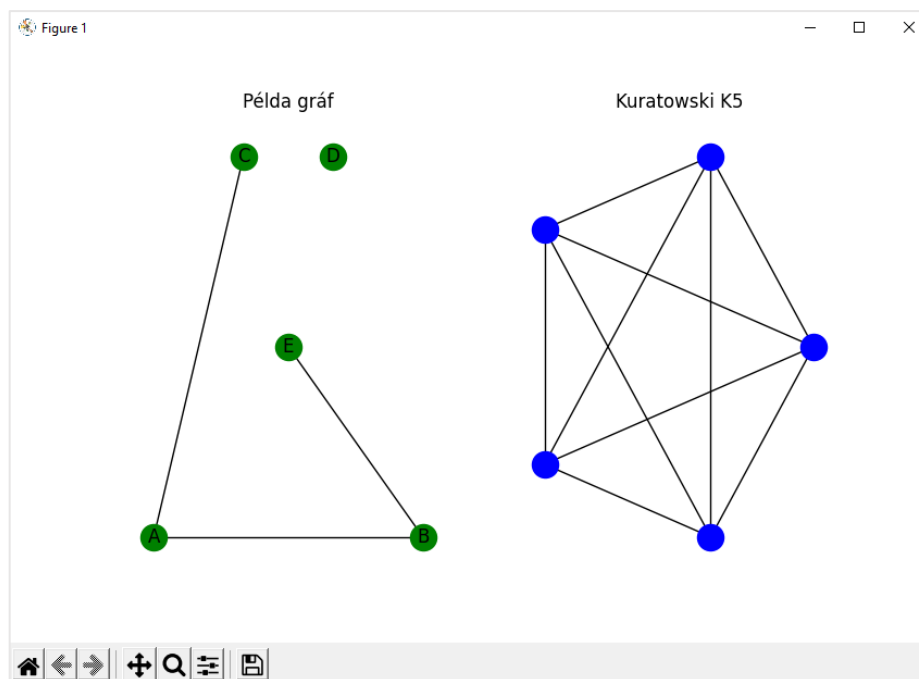
ábra 4.4 - Példa programrészlet `NetworkX` gráfok létrehozására. `P` gráf öt csúcsból és három élből áll, míg `K` gráf a teljes Kuratowski K_5 gráfot reprezentálja.

A létrejött gráfok ezek után módosíthatóak, kezelhetőek vagy megjeleníthetőek. Jelentősen megkönnyíti a hálózatokkal való munkát a `NetworkX`, főleg ha több száz vagy több ezer ponttal dolgozik az ember.

4.1.2.2 Gráf kirajzolása

A létrehozott és módosított gráfokat képesek vagyunk megjeleníteni a `Matplotlib` `Pyplot` nevű alkönyvtárával [21]. Alapvetően egy `Matlab`-hoz hasonló függvény kirajzoló eszköz, azonban a `NetworkX`-ben definiált és felvett adatokat is könnyedén képes megjeleníteni.

Többféle megjelenítési mód van a palettán, azonban a legegyszerűbb és legátláthatóbb módszer, ha definiáljuk a csúcsok fix pozícióit a képernyőn. Így minden alkalommal azonos helyre kerülnek felvételre a gráf csúcsai és élei. Lehetőség van az élek és csúcsok színeinek és formáinak is a megváltoztatására, illetve különböző címkékkel lehet ellátni a gráfot, a könnyebb értelmezés érdekében.



ábra 4.5 - Példa megjelenítés az előzőekben létrehozott gráfok alapján.

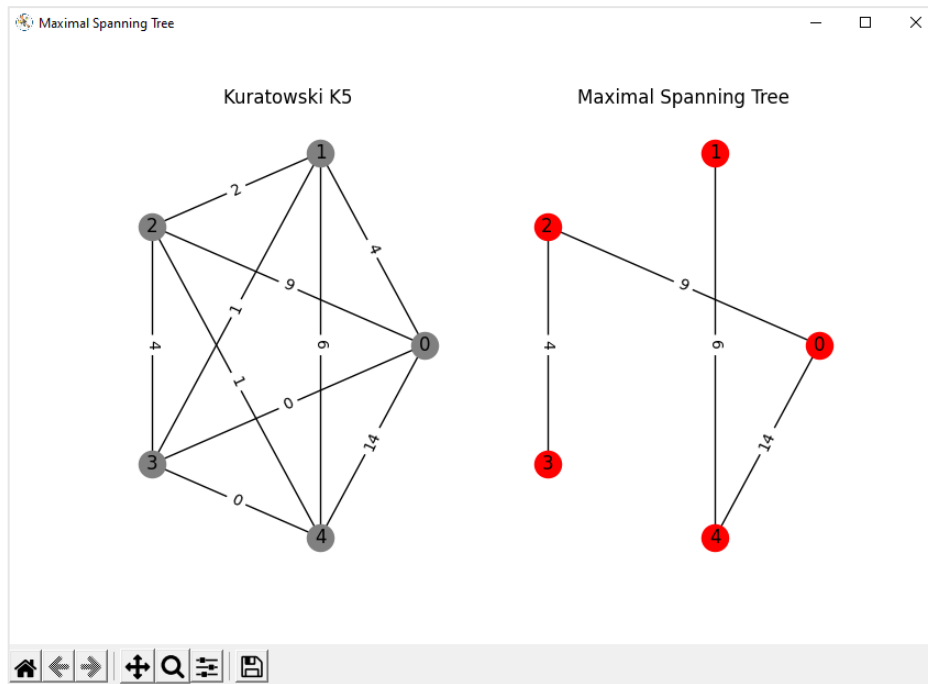
4.1.2.3 Különböző gráfalgoritmusok

A program megalkotása során felhasználtam néhány előre definiált metódust is, amelyek működéséről szeretnék röviden szót ejteni.

A *maximal_spanning_tree* metódus [22] egy felvett gráfra kiszámolja a maximális súlyú feszítőfát. Fontos szempont, hogy a súlyok előre legyenek definiálva és lényeges, hogy eltérő súlyok legyenek a gráf élein. A függvény több lehetséges gráfalgoritmus közül képest választani, amelyek visszaadják a feszítőfát. Ezek lehetnek a Kruskal, a Prim és a Borůvka algoritmusok. A feladat megoldása során a Kruskal algoritmusú (lásd: 3.2.1.2) verziót alkalmaztam.

Az előző példák alapján szemléltetném, a maximális súlyú feszítőfa működését az alábbi ábrán (ábra 4.6). A K_5 gráf éleire random módon súlyokat helyeztem 0 és 15 közötti egész számokkal. A kialakult feszítőfa mindig a maximális összsúlyt keresi addig, amíg a K_5 gráf még körmentes.

A következő metódus, amit alkalmaztam a *shortest_path* nevű algoritmus [23]. Ez a Dijkstra- vagy Bellman-Ford-algoritmusok felhasználásával találja meg a legrövidebb utat egy irányítatlan gráfban. Fontos megadni a kezdő és a végpontot, amik között a legrövidebb utat keressük. Ezt a funkciót az access switch-ek közötti utak felderítésére és az élek súlyainak kialakítására használtam.



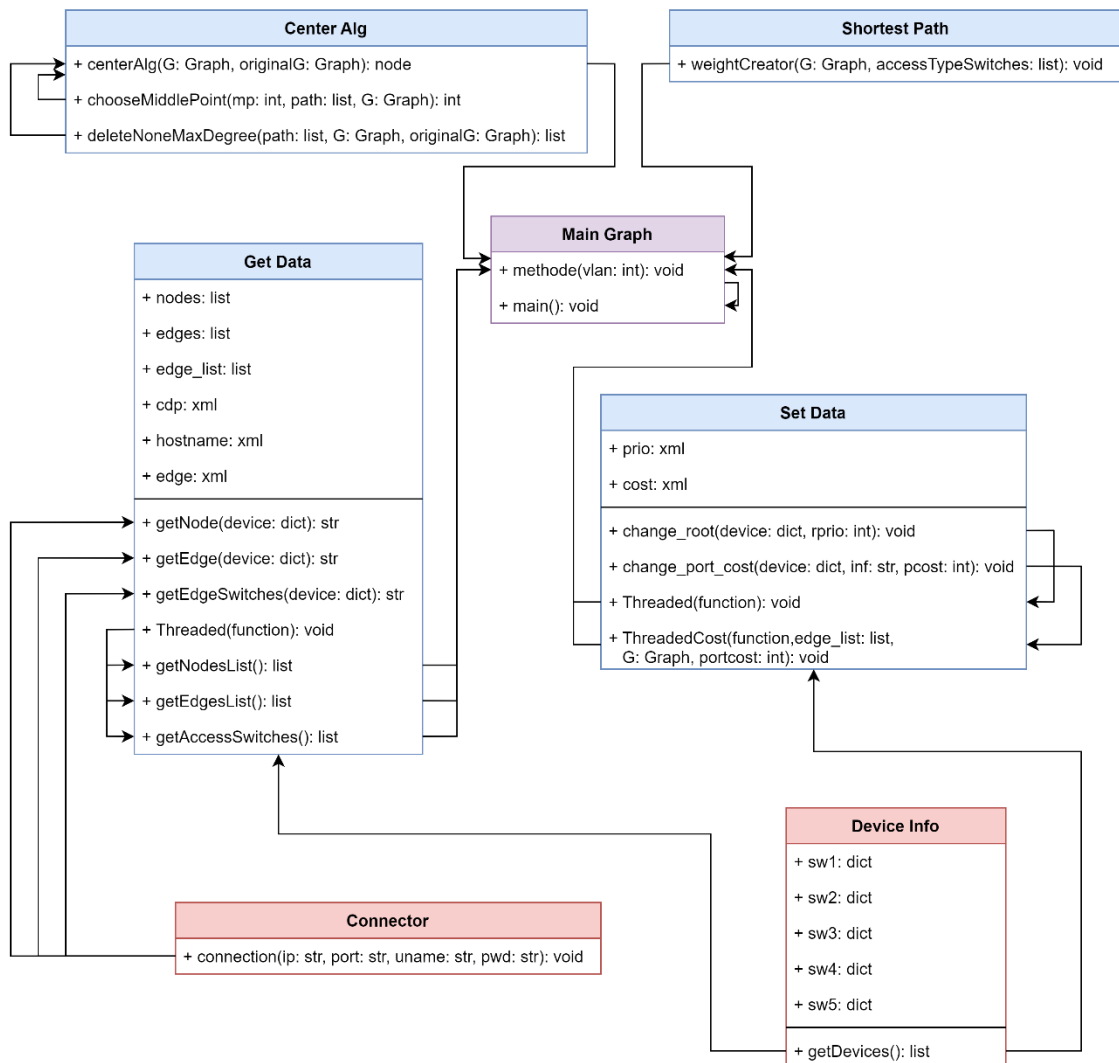
ábra 4.6 - Maximális súlyú feszítőfa példája. Kuratowski K_5 gráf élei random súlyokkal ellátva (balra), a *maximum_spanning_tree* algoritmus eredménye (jobbra).

Az *all_simple_path* segédfüggvényt a kialakult fa középső elemének megkeresésére használtam. [24] Ugyanis a középső elem a fában lévő leghosszabb út középső eleme(i). Sajnos a NetworkX csomag nem tartalmazott leghosszabb útvonalat kiszámoló függvényt, ezért az összes út közül választottam ki a leghosszabbat. Például az ábra 4.6-on látható kialakult feszítőfa esetén az összes *all_simple_path* algoritmus eredményei közül a leghosszabb az $\{1,4,0,2,3\}$ útvonal lenne.

Utolsó megemlíthető funkció a csúcsok fokszámaival kapcsolatos. A NetworkX gráfokban lehetőség van lekérdezni az adott csúcsokhoz tartozó fokszámot. Ez akkor tud előnyös lenni, ha például a csúcs szomszédjainak számára vagyunk kíváncsiak. Ha létrejött egy G gráf, akkor egy adott csúcshoz tartozó fokszámot a következőképpen tudunk lekérdezni: $G.degree[N]$.

4.2 A program működése

Ebben a részben bemutatom a program felépítését és a fontosabb funkciókat, amelyek az új feszítőfa-beállításokkal foglalkoznak. Első sorban szeretnék egy átfogó képet alkotni a program felépítéséről, amely a lenti ábrán látható (ábra 4.7). Az elkészült program több kisebb alprogramból épül fel, amelyek egymás függvényeit elérve és alkalmazva alakítják a hálózatot.



ábra 4.7 - A program felépítésének rajza. Az alprogramokban található attribútumok és függvények eltérő helyen való alkalmazásait a nyilak végpontjai jelzik.

Külön jelöléssel láttam el az eltérő funkcióval rendelkező alprogramokat. A lila színű *Main Graph* nevű modul a lelke a programnak, innen hívódnak meg megfelelő logikai sorrendben a különböző függvények és itt történik az eredmények kirajzolása is.

A piros színű alprogramok az eszközökhöz való csatlakozást biztosítják, illetve tárolják a csatlakozási adatokat.

A kék színű részek pedig a különböző változtatások és számítások elvégzéséért felelősek. Ezen alprogramok kéri el a switch-ektől az adatokat a gráf felépítéséhez, számolják ki az új feszítőfát és a legvégén érvényre is juttatják a megváltoztatott paramétereket.

Az átfogó kép kialakítása után ismertetem az egyes részeket részletesebben a mélyebb megértés végett.

4.2.1 Main Graph

A főprogram felelős az egyes részek összekötéséért, illetve itt történik meg a felvett gráf módosítása, alakítása és az eredmények kirajzolása is. A program első lépésként létrehoz egy üres gráfot a NetworkX részben leírtakhoz hasonlóan. Ezután a csúcsok felvétele kezdődik el, amit a *Get Data* alprogram *getNodeList()* függvénye indukál és adja vissza az eszközök sorszámát a nevük alapján.

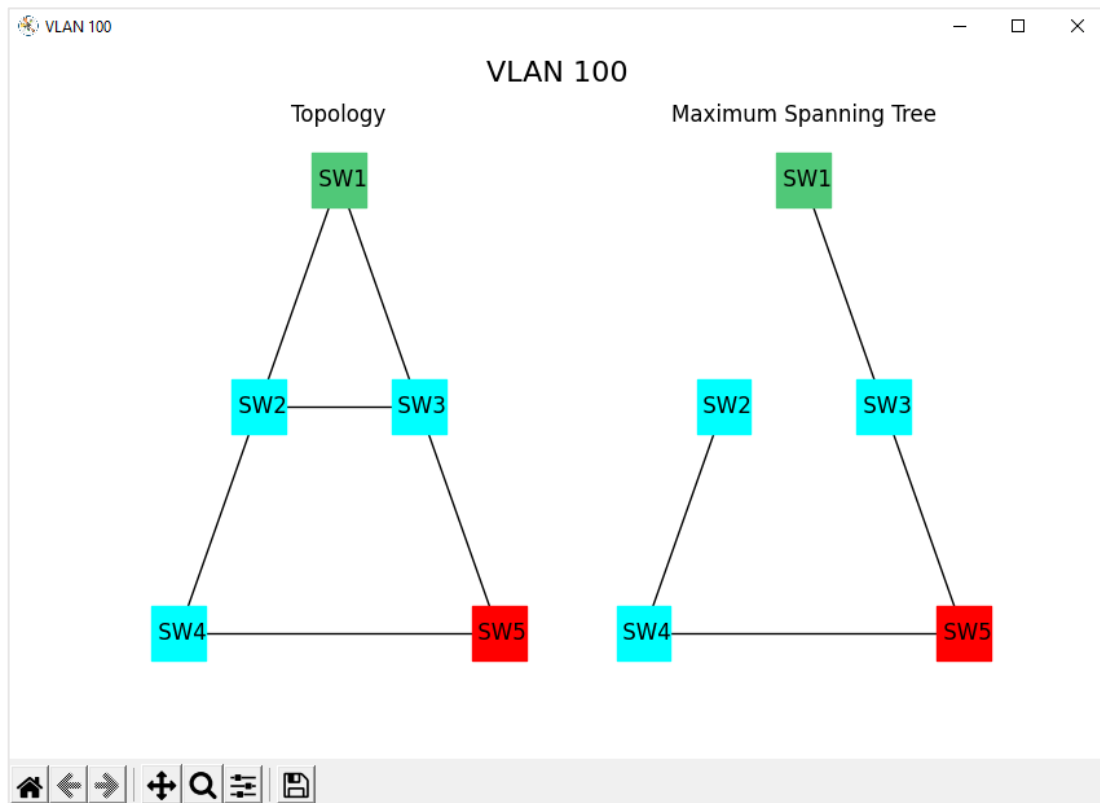
Hasonlóképpen történik a következő lépésben az élek felvétele a *getEdgesList()* metódus meghívásával. Itt a CDP szomszédok alapján történik az élek felvétele. Mind a kettő lépésnél egyesével belép a program az eszközökbe az adatok lekérése érdekében, azonban szálanként elkülönítve (lásd 4.2.8).

A kialakult gráf után még mindig a *Get Data* modult használva a főprogram elkéri az access switch-ek listáját (*getAccessSwitches()*) és eltárolja egy listában. Az eltárolt switch-ek ezután a *Shortest Path weightCreator()* nevű függvényéhez kerülnek átadásra. Itt alakulnak ki a gráfon az új élsúlyok, amivel már meg lehet határozni a maximális súlyú feszítőfát az aktuális gráfra.

A kialakult feszítőfára ezután a *centerAlg()* függvényt alkalmazza a program, ami visszaadja a középső elemét a fának, így a Root Bridge szerep is meghatározásra került. Minden adat birtokában pedig a *Set Data* alprogram függvényeivel a változások érvényesíthetők a topológián. Úgyszintén ezek a változtatások is párhuzamosítva történnek meg.

A *Main Graph* érdekessége még az aktuális hálózat és a kialakított feszítőfa megjelenítésének a képessége. A felvett gráf és a fa alapján megjeleníthetők az eredmények a Matplotlib könyvtár segítségével. Különböző grafikai beállításokat

alkalmazva elkülönítettem a megjelenítés során az eltérő típusú switch-eket színek alapján. A gyöker kapcsoló piros színnel került megjelenítésre, míg az access switch-ek zöld színnel. A többi switch, amelyeknek nincsen speciális szerepük a hálózatban, kék színt kaptak. Ezen kívül az eszközök neveit is megjelenítettem a csúcsokon, az azonosíthatóság végett, illetve az eszközök közötti kapcsolatokat is jelöltem. Így látható a különbség a valós összeköttetés és a kialakult fa felépítése között. (ábra 4.8)



ábra 4.8 - A hálózat kirajzolása. Piros switch = root bridge, zöld switch = access switch, kék switch = nincs különleges funkció.

A *main()* függvényből indítható a *methode(vlan)* függvény több különböző VLAN-ra. Itt is igyekeztem megoldani a szálanként való elkülönítést, azonban a Matplotlib megjelenítés miatt (ami csak a fő szálon futhat) nem tudtam ezt megvalósítani. Kirajzolás nélkül viszont több eszközhöz, több VLAN-ra, egyidőben képes lenne a program az átalakítások elvégzésére.

4.2.2 Device Info

Az eszközökhöz való csatlakozáshoz szükséges az elérési adatokat eltárolni egy egyszerű struktúrában. A kapcsolathoz szükséges az eszköz elérési IP címe, a csatlakozási port (22 - SSH vagy 830 - TCP over SSH [NETCONF]), a belépési felhasználónév és

jelszó. Ezeken kívül pedig még eltároltam az eszközök neveit könnyebb azonosítás miatt. Az elérési adatok egy szótár (dictionary) struktúrában vannak eltárolva szöveggént. Az egyes szótárakat az alprogram *getDevices()* nevű függvényével lehet elkérni, ami visszaadja egy listában tárolva az összes szótárat.

```
#.
#.
# SW5 IOSvL2 Switch
sw5 = {
    "address": "198.18.1.105",
    "netconf_port": 830,
    "ssh_port": 22,
    "username": "cisco",
    "password": "cisco",
    "name": "SW5"
}

def getDevices():
    return [sw1,sw2,sw3,sw4,sw5]
```

ábra 4.9 - Az egyes eszközök csatlakozási adatainak tárolása, illetve a szótárak lekérési függvénye.

4.2.3 Connector

A *Connector* rész az ncclient manager objektumának „connect” függvényét valósítja meg [25] csak általánosabb kivitelben. A többszöri felhasználás érdekében került kiszervezésre egy külön függvénybe. A *connection* függvény leegyszerűsíti a kötelező adatok megadását (IP-cím, port szám, felhasználónév és jelszó). Ezeket az adatokat az egyes függvények mind a *Device Info* modultól szerzik meg.

```
from ncclient import manager
import sys

def connection(ip,port,uname,pwd):
    try:
        conn = manager.connect(host=ip, port=port, username=uname, password=pwd,
                               hostkey_verify=False,look_for_keys=False)
        return conn
    except (TimeoutExpiredError,SSHError):
        print("Connection timed out or could not be established!")
```

ábra 4.10 - Connection függvény felépítése. Az ncclient.manager.connect függvény alapján

4.2.4 Get Data

A gráf felépítéséhez és az access switch-ek felvételéhez használt függvények mindegyike azonos struktúra alapján működik. Először az előző részben leírt *connection* függvény SSH kapcsolatot épít ki az eszközzel, majd egy előre definiált RPC üzenetet küld a switch-nek. Fontos megemlíteni, hogy a program megalkotása során küldött RPC-k eltérnek a YANG adatmodellben megfogalmazott feltételektől. Az számomra elérhető eszközök nem támogatták a különböző YANG modelleket, ezért az aktuális parancssori utasítást küldi el a kontroller. Erre a következő példa (ábra 4.11) adhat egy jó szemléltetést, az üzenet felépítéséről. Természetesen a NETCONF szabványban is meghatározott módon, XML formátumba csomagolva haladnak az üzenetek.

```
edge = '''
    <filter>
        <config-format-text-cmd>
            <text-filter-spec> | include hostname</text-filter-spec>
        </config-format-text-cmd>
        <oper-data-format-text-block>
            <exec>show spanning-tree vlan %s</exec>
        </oper-data-format-text-block>
    </filter>
    '''
```

ábra 4.11 - Szemléltető ábra parancssori utasítást küldő RPC üzenetre. Az <exec> blokkba tartozó parancs fut le az eszközön, ha helyes a formátuma.

A visszaérkező RPC-reply üzenet tartalma a szöveghalmaz, amit az eszköz alapból is kiírna válaszként a CLI felületre. Ezért egy kicsit nehezebb módon kell a megfelelő adatot kinyerni a szövegből. Ez történhet a szöveg feldarabolásával vagy XML formátummá való alakítással is. A kívánt adatokat ezután el lehet tárolni vagy továbbküldeni más programrészeknek.

A *getNode* függvény egyszerűen az adott eszköz nevét kéri el és tárolja egy listában, a *getEdgeSwitches* függvény pedig az STP információk közül keresi azokat az eszközöket, amelyeknek az adott VLAN-ban létezik „P2P Edge” mező (ábra 3.3).

Az élek felvételénél nem csak a két csúcs között húzódó él kiléte a fontos, hanem az összeköttetéshez használt port-ok sorszámai is. Ez a későbbi port költségek változtatásánál nyújt szerepet. A CDP információk alapján eldönthető a szomszédosság, illetve az összeköttetésre használt port-ok is, ezért az *getEdge* függvény ezt a kimenetet vizsgálja. Egy adott élhez a NetworkX-ben lehetőség van különböző extra attribútumokat

definiálni. Minden egyes élhez kettő attribútumot definiáltam: *port1* és *port2*. Az első mindig az aktuálisan vizsgált switch interfészének a száma, a második pedig a szomszédé. Például a 2-es és a 4-es eszközök között húzódó élhez az alábbi információk kerülnek felvételre:

```
(2, 4, {'weight': 0, 'port1': '2 Gig0/0', 'port2': '4 Gig0/3'})
```

A felvétel pillanatában az összes él 0 súlyt kap, ami később módosítható lesz. Ezzel pedig az összes olyan adat rendelkezésre áll, amellyel megkezdhető az új fa kialakítása.

4.2.5 Shortest Path

Ezen alprogram tartalmazza azt az egyetlen függvényt, amely az élek súlyainak módosításával foglalkozik. A *weightCreator* függvénynek szüksége van a teljes gráfra, hogy elérje az csúcsait és az éleit, emellett pedig a már létrehozott access switch listára. Minden közvetlen végkapcsolattal rendelkező switch között megvizsgálja a függvény a legrövidebb utakat és ezeket eltárolja egy listában.

A következő lépés ezen a listán való végigiterálás és közben minden olyan él súlyának növelése, amelyik él létezik az utak listájában. A súly növelésének is megvan a maga szabályrendszere (táblázat 3.1). Alapvetően a gráfban lévő él két végpontjának a fokszámmösszegeivel nő a súly, ha pedig az út hossza egységnyi, akkor egy konstans számmal tovább nő a súly értéke.

```
import networkx as nx

def weightCreator(G, accessTypeSwitches):
    n = len(accessTypeSwitches)
    sumpath = int((n*(n-1))/2)
    w = []
    for i in range(0,n):
        for j in range(i+1,n):
            sp = nx.shortest_path(G,accessTypeSwitches[i],accessTypeSwitches[j])
            print("Shortest path between: "+str(accessTypeSwitches[i])+" and "+str(accessTypeSwitches[j])+
                  " :"+str(sp))
            w.append(sp)
    for paths in w:
        if(len(paths)!=0):
            for n in range(0,len(paths)-1):
                actual_weight = G[paths[n]][paths[n+1]]['weight']
                node_degrees = G.degree[paths[n]] + G.degree[paths[n+1]]
                listofNeighbours = G.neighbors(paths[0])
                adj_adder = 0
                if paths[len(paths)-1] in listofNeighbours:
                    adj_adder = 10
                G[paths[n]][paths[n+1]]['weight'] = actual_weight + node_degrees + adj_adder
```

ábra 4.12 - Súly változtatására írt függvény. Az első iteráció során a legrövidebb utak felvétele valósul meg, a második során pedig a súlyok változtatása a felállított szabályrendszer szerint.

Fontos megjegyezni, hogy a függvénynek nincsen visszatérési értéke (void), mivel a súlyok változtatása a gráfhoz kötött éleken közvetlenül változásra kerül. Az így kialakult új súlyokra már alkalmazható a maximális súlyú feszítőfa-algoritmus.

4.2.6 Center Algorithm

Két ok miatt döntöttem úgy, hogy a kialakult fa középső elemét meghatározza a módszer. Először is a gyökér a hálózat kialakítása szempontjából előnyös, ha a rendszer közepén van. A másik ok pedig a megváltoztatott Root Bridge helye a hálózatban nagyban megkönnyíti az új élek elrendezését. A 4.1.2.3-ben leírt módon a középső elem megkeresése a leghosszabb út középső elemének vagy elemeinek a megtalálásában rejlik. Ha a leghosszabb úthossz páros, akkor a sor mediánja mindig kettő elem kell legyen, páratlan hossz esetében egy csúc a középső elem.

Ha kettő centerben lévő switch lett az eredmény, akkor a következő módon döntöttem el az előnyösebb kiválasztását. Ha az elsőnek nagyobb vagy egyenlő a fokszáma, akkor a program azt választja, ellenkező esetben a másodikat.

Az STP sajátossága miatt pedig minden Root Bridge eszközön lévő aktív portok designated port-státuszban kell legyenek. Ez azt jelenti, hogy a középső elem csak olyan switch lehet, amelynek minden port-ja továbbítja az adatszegmenseket. Ezt a problémát úgy oldottam meg, hogy minden egyes csúcsot töröltem a leghosszabb út listájából, aminek az eredeti gráfhoz képest nem azonos a fokszáma. Ez a függvény a *deleteNoneMaxDegree* a program struktúrájában, aminek felépítése megfigyelhető a következőkben is (ábra 4.13).

```
def deleteNoneMaxDegree(pathlist,G,originalG):
    del_list = []
    for l in pathlist:
        if G.degree[l] == originalG.degree[l]:
            del_list.append(l)
    return del_list
```

ábra 4.13 - A nem maximális fokszámú pontok törlésére szolgáló függvény.

4.2.7 Set Data

A *Set Data* modul két hálózati paraméter megváltoztatásáért felelős, amelyek a root azonosító és a port költség. A *Get Data* alprogramhoz képest kevés lényegi eltérés van a megvalósításban. Az egyik ilyen különbség, hogy a küldött RPC üzenetek utasítástípusa <edit-config>, ami lehetővé teszi az switch-ek számára a kívülről kapott parancsok érvényre juttatását.

Ugyancsak a modulnak ismernie kell a switch-ek csatlakozási paramétereit, majd a *connection* metódus segítségével ki tudja építeni a biztonságos kapcsolatot. A változtatások indukáló függvényeknek szükségük van a változtatandó paraméterekre, amelyeket a főprogramban való meghíváskor kapnak meg.

A beállítás változtató függvények rendkívül modulárisak és szinte azonnali változást biztosítanak az eszközökön. Ezért több helyen vagy akár több programban is felhasználhatóak lehetnek. A külön eszközökbe való belépés és módosítás itt is szálanként elkülönítve történik a lefutás gyorsítása végett.

A NETCONF <edit-config> utasítása többféle konfigurációt képes változtatni a Cisco eszközökön [26]. A különböző konfiguráció módok eltérő funkciót látnak el az eszközök működése során. NETCONF segítségével ki meg lehet adni a változtatni kívánt konfiguráció típusát, amelyek a következők lehetnek:

- **Running config:** direkt konfigurációmódosítást tesz lehetővé.
- **Startup config:** lehetőséget biztosít az eszköz leállítása után a konfiguráció elmentésére.
- **Candidate config:** A running config másolata, amelyet később érvényre lehet juttatni.

Ebben a modulban található függvények mindegyike a *running* konfigurációt módosítja (ábra 4.14) és szinte azonnal életbe is lépnek a változások.

Az élek felépítésekor eltárolt port számoknak a *change_port_cost* függvényben lesz értelme, hiszen a költségek változtatásához szükséges, hogy melyik eszköz mivel van összekötve, melyik interfészen keresztül. Tehát a függvény először a *port1* paraméterben lévő interfészen állít új költséget, majd az él másik oldalán lévő port-on is így tesz csak *port2* paraméter értékével.

```

def change_root(device,rprio,vlan):
    try:
        string_prio = prio % (str(vlan),rprio)
        with connection(device["address"],
                        device["netconf_port"],
                        device["username"],
                        device["password"]) as devicem:
            reply=devicem.edit_config(target='running',config = string_prio)
            if(devicem.ok):
                print("Updated priority on "+str(device['name'])+" to "+rprio)
    except (ncclient.operations.errors.TimeoutExpiredError,
            ncclient.transport.errors.SSHError):
        print("Connection timed out or could not be established!")

```

ábra 4.14 - Az <edit-config> utasítás célpontja: running. A megfelelő beállításokat a függvény az aktuálisan futó konfigurációban cseréli ki.

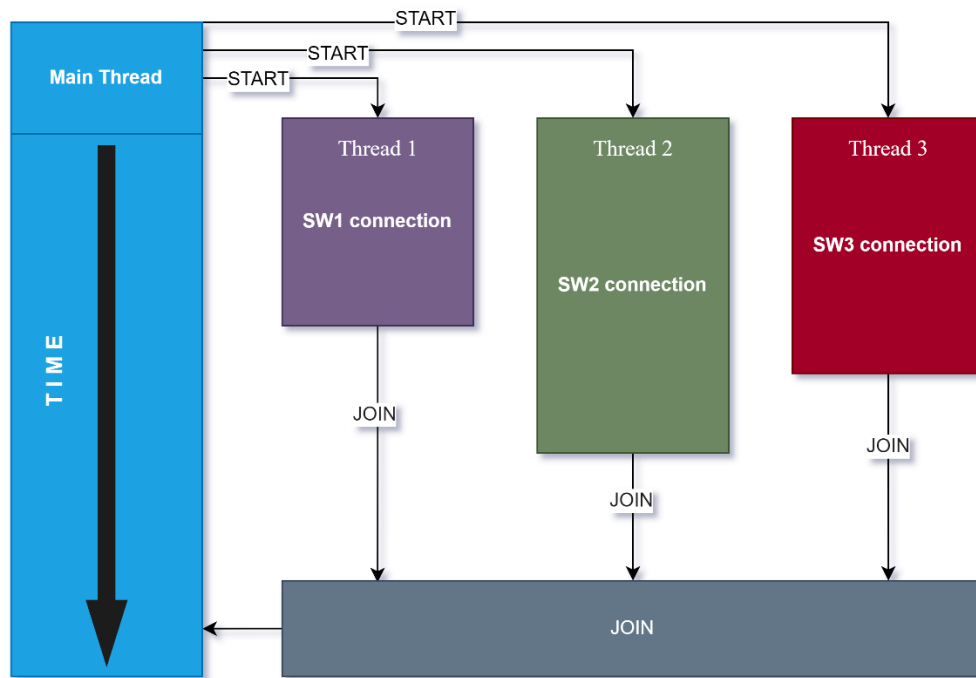
4.2.8 Szálkezelés

A szinkron programozási nyelvek (mint például a C++, Java vagy Python) lineáris lefutást megvalósítva egymás után hajtják végre az utasításokat. Ez a szisztéma megfelelő, ha az egymás után következő feladatokat kell megoldjon a program. Eltérő esetben, ha egy időpontban több feladatot párhuzamosan kellene végrehajtani, akkor használhatóak az elkülönített szálak (thread).

A szálak több lefutási sávot biztosítanak egymás mellett elhelyezkedve, így egy adott feladat elvégzésére (ami több időt igényel) nem kell várjon a program többi része. Az eddigiekben bemutatott program esetében nagyon időigényes művelet az SSH kapcsolat kiépítése. Ennek kezelésére, az eszközökhöz való csatlakozást megvalósító függvényeket mind külön szálakhoz rendelttem. Tehát a program egyidőben képes több switch-be belépni, ott elkérni vagy módosítani az adatokat, majd a program lefutása folytatódhat a megszokott módon.

A nyelvek többsége alapvetően támogatja a szálkezelést, ez a Python esetében is így van. Létezik egy *threading* nevű könyvtár [27], ami tartalmazza azokat a segédfüggvényeket, amelyek a szálakat megnyitják, kezelik majd leállítják. Fontos szempont, hogy a külön szálon futó feladatok bevárják egymást, még a megszűntetésük előtt, elkerülve az egyes hibák kialakulását.

A következő ábrán (ábra 4.15) megfigyelhető a szálak indítása az egyes eszközök csatlakoztatása érdekében, majd a szálak egyesítése és a főszálhoz való csatlakozás és látható.



ábra 4.15 - Szálkezelés szemléltetése. Az egyes szálakat a főszál indítja, majd egymás bevárása után visszatérhet a folyamat a főszálba.

4.3 A program futtatási eredményei

Ez a fejezet írja le az elkészült program futtatási eredményeit és a kiépített virtuális hálózat rendszerét. A megvalósítás során a Cisco CML (Cisco Modelling Labs) virtuális hálózati környezetet használtam. Itt valósítottam meg azt a példa hálózatot, amit valós eszközökön is létre lehetne hozni. A futtatás szemléltetésére néhány eseten is szeretnék végighaladni, a program funkciójának átadása érdekében.

4.3.1 A virtuális hálózati környezet bemutatása

A Cisco nagyvállalat DevNet csapata szabadon elérhető és foglalható virtuális hálózatokat biztosít a SandBox rendszerében¹. Több különböző hálózati téma között (IoT, Networking, Data Center) megtalálható a CML (Cisco Modeling Labs) is, ami egy eszköztárat biztosít virtuális felületen, eszközök konfigurálására és menedzselésére.

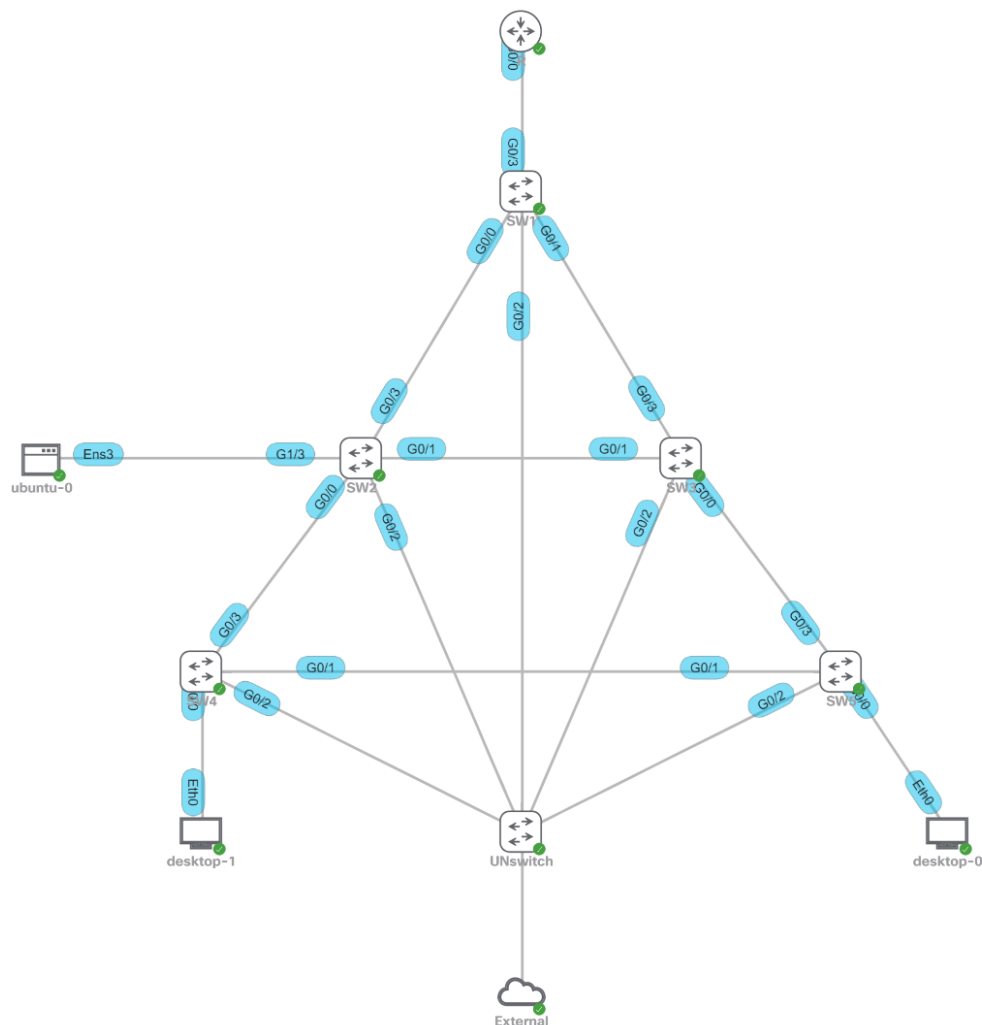
A virtuálisan futó berendezések funkciói majdnem teljesen megegyeznek egy valós rendszerrel, így ideális és költséghatékony különböző fejlesztések vagy tesztek használatára. Az általam elkészített program működésének vizsgálatára is ezért választottam. Minden alkalmazott switch-en IOSv operációs rendszer van használatban.

¹ <https://developer.cisco.com/site/sandbox/>

A SandBox felület a lefoglalt időkereten belül VPN kapcsolatot biztosít a CML rendszerhez. A kialakult VPN kapcsolat során egy alhálózatba kerülnek a CML-ben hozzáadott eszközök és a kontroller, ahol a program fut.

Az STP működése érdekében egy egyszerű topológiát alakítottam ki, majd minden szükséges beállítást elvégeztem a switch-eken, hogy a hálózat megfelelően tudjon üzemelni. Megfelelő összeköttetések, IP-címek beállítása, Rapid PVST+ protokoll engedélyezése az eszközökön és a teszteléshez használt VLAN-ok helyes beállítása.

A switch-eken kívül lehetőség van végfelhasználó eszközöket, virtuális operációs rendszereket felvenni. Ez a használat során Linux alapú rendszereket jelent, amelyek reprezentálják a hálózatot használó emberek személyi számítógépeit. Az ábra 4.16 jelöli a kialakított topológiát, ahol az eddig bemutatott eszközökön kívül még fellelhető egy router, a VLAN-ok közötti útvonalválasztások végett, illetve egy külső csatlakozó (External Connector), ami a VPN hálózathoz való kapcsolódást biztosítja.



ábra 4.16 - CML rendszerben kialakított hálózati topológia.

4.3.2 Eredmények

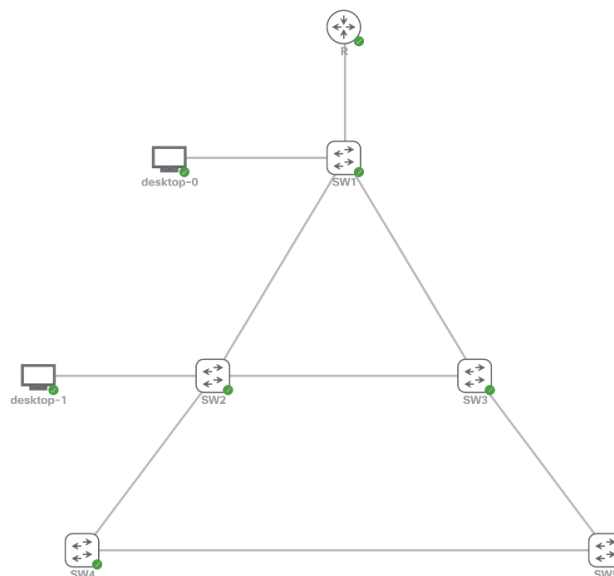
A program működésének szemléltetése érdekében kettő eltérő esetet szeretnék bemutatni. Az elsőben csak egy VLAN-ra történő futtatási eredményeket szeretném részletesebben bemutatni, az átláthatóság miatt. A második esetben pedig több VLAN-t használó hálózati változtatást szeretnék átfogóan szemléltetni.

Egy helyzethez több kialakult feszítőfát is megjelenít a program, ami a későbbi kiértékelés és az egymáshoz való viszonyítás érdekében történik. Ezek a feszítőfák a következők:

- Az alapállapotban lévő feszítőfa, STP beállítások **változtatása nélkül**.
- A MAC-címek alapján növekvő sorrendben felvett csúcsok alapján és nulla élsúllyal ellátott **minimális** súlyú feszítőfa.
- Az általam felállított módszer alapján kialakult **maximális** súlyú feszítőfa.

4.3.2.1 Első eset

Az első esetben a 100-as számú VLAN-ban lévő eszközök mozgása után kialakult új feszítőfa átalakítása játszik szerepet. Alapfelállásban (ábra 4.17) kettő számítógép csatlakozik a hálózathoz, *SW1* switch-hez *desktop-0* számítógép és *SW2* switch-hez *desktop-1* számítógép. Más VLAN-ba tartozó eszköz nincs a hálózatban és a switch-ek és PC-k közötti kapcsolat beállításai is helyesek.

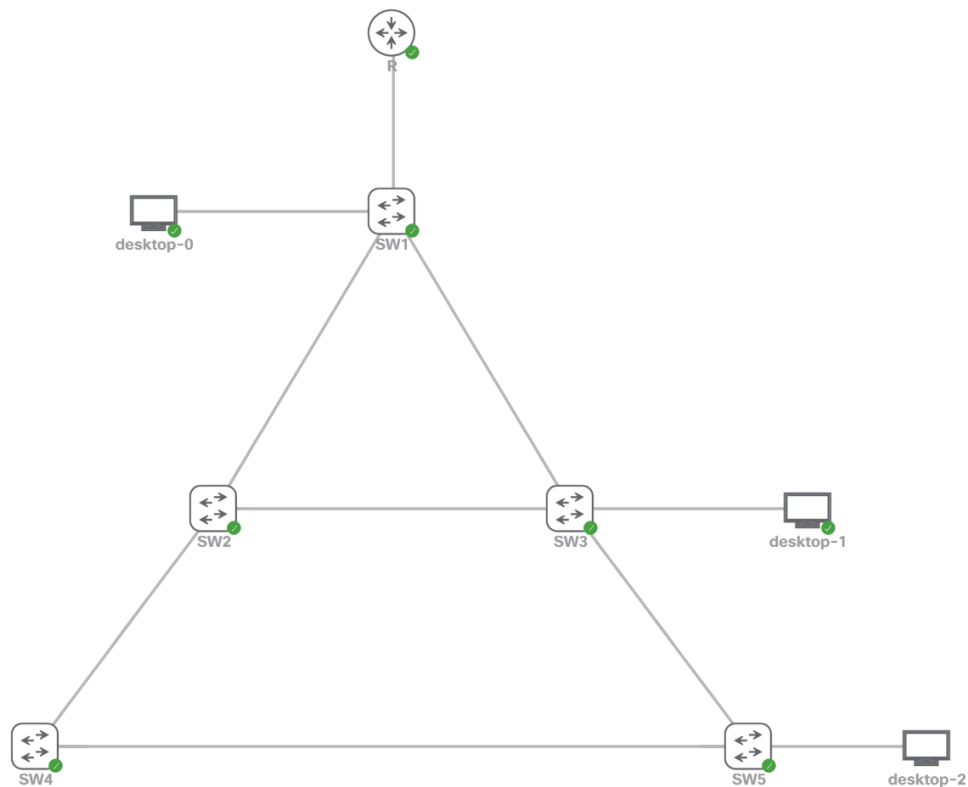


ábra 4.17 - Kiinduló állapot felépítése. *Desktop-0* és *desktop-1* VLAN 100-ban lévő végponti eszközök csatlakoznak a hálózathoz.

Az eddig részletezett módon egy fiktív vállalati környezetben a számítógépek felhasználói új helyet foglaltak el az épületen belül, ezért új switch-en keresztül képesek a hálózathoz csatlakozni. Ezen felül egy új kolléga is érkezett a céghez, aki szintén elfoglalja helyét az irodában.

Switch	Alapfelállásban csatlakoztatott eszköz	Új felállásban csatlakoztatott eszköz
SW1	Desktop-0	Desktop-0
SW2	Desktop-1	-
SW3	-	Desktop-1
SW4	-	-
SW5	-	Desktop-2

táblázat 4.2 - Csatlakozott eszközök változásának összefoglalása.



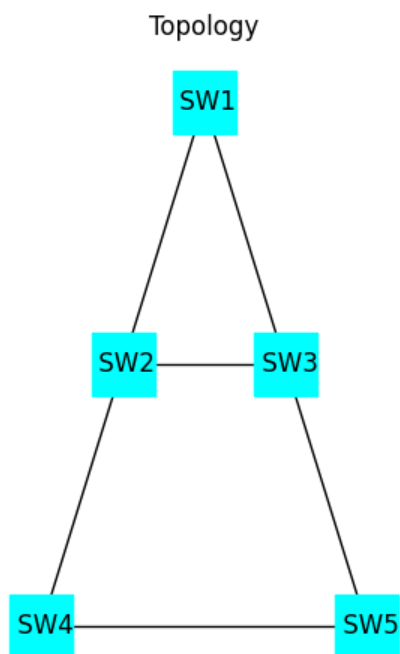
ábra 4.18 - Megváltozott új állapot felépítése. *Desktop-0* helye változatlan, *desktop-1* SW3 switch-hez csatlakozott, *desktop-2* új felhasználó SW5-höz csatlakozott.

Az új felállítás beállta után indítható a program, ami kiszámolja és átrendezi az STP beállításokat. A gráf felépítéséhez, az új számítások elvégzéséhez és az eredmények beállításához néhány másodperc elegendő. A hálózat továbbra is használható lesz és a Rapid PVST+ protokoll használata miatt a konvergencia idő is alacsony.

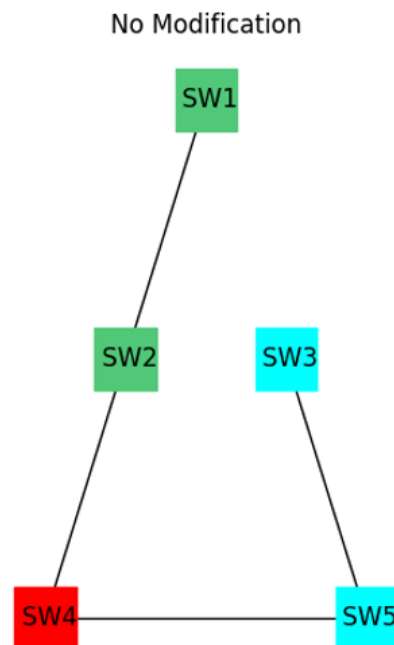
A program futása végén négy rajz jelenik meg a képernyőn, ami a hálózatot és a kialakult feszítőfák alakulását reprezentálja. Az első egy egyszerű gráf képe, amit a program a switch-ek alapján felvett és kirajzolt. A második az alapfelállításban lévő feszítőfa képe, ez viszonyítási pontként szerepel az eredmények között. A harmadik kép egy sima minimális súlyú feszítőfa algoritmus eredménye, ahol csak a fa alakja változik, egyéb beállítások nem. Az utolsó ábrán pedig az általam készített program eredménye figyelhető meg.

Az ábrák színezése a különböző switch-szerepeket jelölik a hálózatban. Kék az egyszerű switch, ami részt vesz a forgalom továbbításában és az STP-ben, de kitüntetett szerepe nincs. A zöld színnel jelöltek az access switch-ek (amikhez kapcsolódik PC), a piros pedig a Root Bridge-t jelöli.

VLAN 100

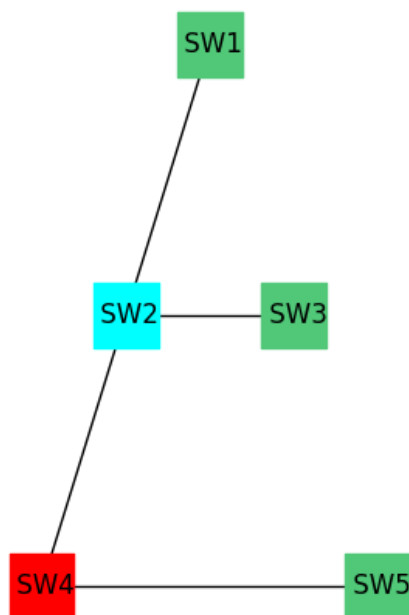


ábra 4.19 - A hálózat alapján kialakított gráf képe.



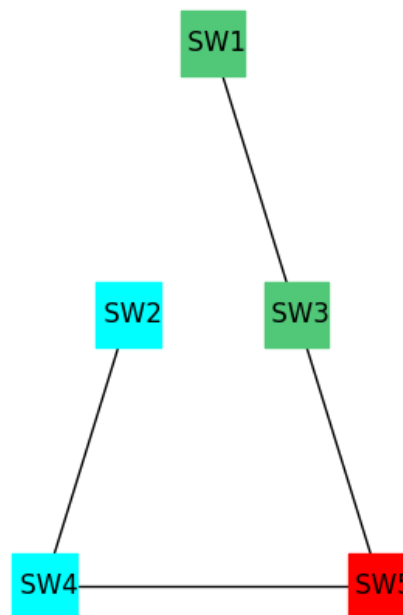
ábra 4.20 - Változtatások nélküli alapfelállásból adódó feszítőfa ábrája.

Simple Minimum Spanning Tree



ábra 4.21 - Egyszerű minimális súlyú feszítőfa
ábrája. Az alapfelálláshoz képest a Root
Bridge változatlan.

Maximum Spanning Tree



ábra 4.22 - Az elkészült program alapján
számolt új feszítőfa ábrája.

A kialakult új feszítőfa (ábra 4.22) beállításai ellenőrizhetők az adott eszközök CLI felületéről is. A fontosabb változásokat elszenvedett switch-ek paraméterei relevánsabbak, ami ebben az adott szituációban az SW5 (hiszen ez az eszköz lett az új gyökérpont) és az SW1, SW2, SW3 eszközök (mert az egyik port-jaik *alternate* státuszúak).

```
SW5#sh spanning-tree vlan 100
```

VLAN0100	
Spanning tree enabled protocol rstp	
Root ID	Priority 4196
	Address 5254.0010.15ed
This bridge is the root	
	Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
Bridge ID	Priority 4196 (priority 4096 sys-id-ext 100)
	Address 5254.0010.15ed
	Hello Time 2 sec Max Age 20 sec Forward Delay 15 sec
	Aging Time 300 sec

Interface	Role	Sts	Cost	Prio.Nbr	Type
Gi0/0	Desg	FWD	4	128.1	P2p Edge
Gi0/1	Desg	FWD	4	128.2	P2p
Gi0/3	Desg	FWD	4	128.4	P2p

ábra 4.23 - SW5 eszközön futtatott STP adatokat megjelenítő parancs (VLAN-100). Látható a root prioritás, illetve hogy ez az eszköz a gyökér. Továbbá minden port-ja *designated* típusú. Ezen felül SW5 nem csak Root Bridge, de access switch szerepet is betölt (Gi0/0).

Interface	Role	Sts	Cost	Prio.Nbr	Type
Gi0/0	Altn	BLK	100	128.1	P2p
Gi0/1	Root	FWD	4	128.2	P2p
Gi1/3	Desg	FWD	4	128.8	P2p Edge

ábra 4.24 - SW1 port státuszok. Gi0/0 alternate-blocking port.

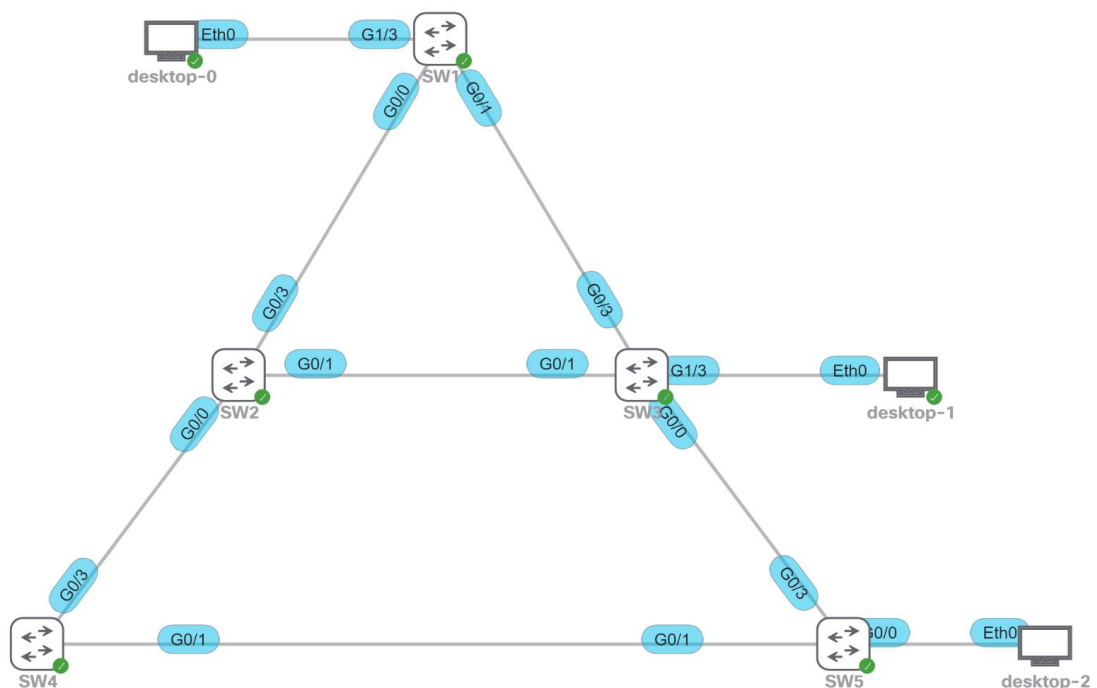
Interface	Role	Sts	Cost	Prio.Nbr	Type
Gi0/0	Root	FWD	4	128.1	P2p
Gi0/1	Altn	BLK	100	128.2	P2p
Gi0/3	Desg	FWD	100	128.4	P2p

ábra 4.25 - SW2 port státuszok. Gi0/1 alternate-blocking port.

Interface	Role	Sts	Cost	Prio.Nbr	Type
Gi0/0	Root	FWD	4	128.1	P2p
Gi0/1	Desg	FWD	100	128.2	P2p
Gi0/3	Desg	FWD	4	128.4	P2p
Gi1/3	Desg	FWD	4	128.8	P2p Edge

ábra 4.26 - SW3 port státuszok. Gi0/1 port designated, azonban az a link ellenkező oldala blocking.

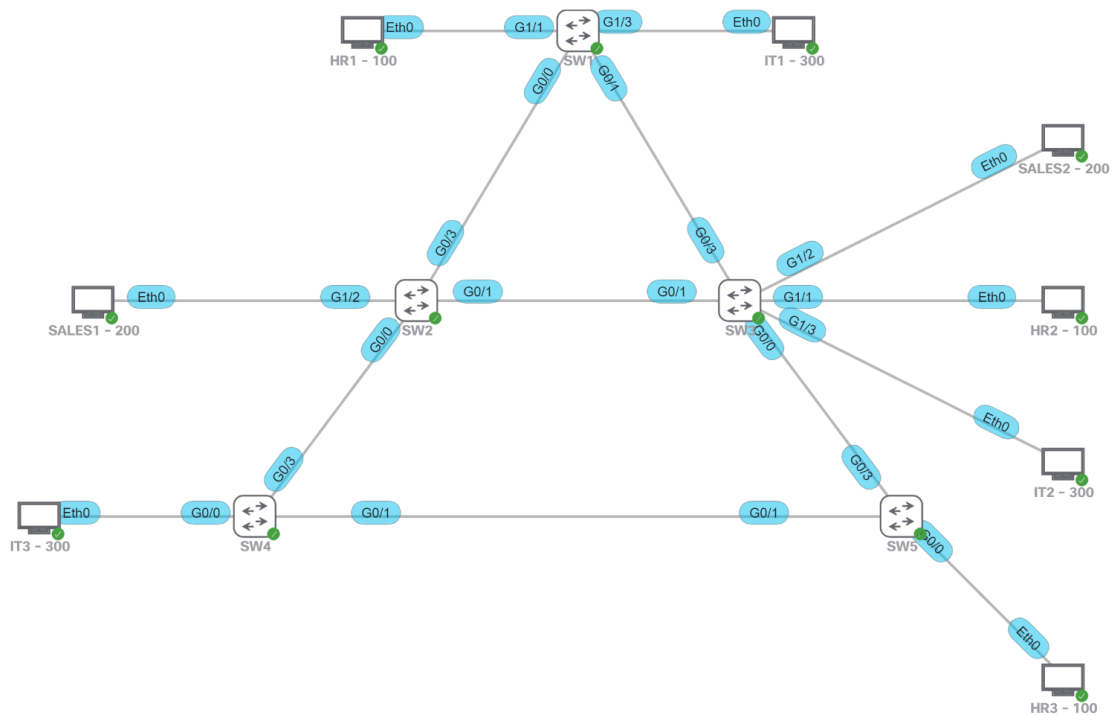
A különböző linkek és interfészek státuszának és szerepének könnyebb megértéséhez az ábra 4.27 segítségével feltüntettem az egyes port-ok neveit. Így már nyomon követhető, hogy mely csatlakozások lettek letiltva a topológiában, illetve melyek vannak engedélyezve, vagy mely irányba mutatnak a root port-ok.



ábra 4.27 - Port azonosítókkal ellátott topológia ábrája.

4.3.2.2 Második eset

A teljes funkcionalitás bemutatása érdekében több VLAN esetében is lefuttattam a programot. A fiktív vállalatban ebben az esetben három munkacsoport eszközei vannak külön VLAN-okba elkülönítve (HR, Sales és IT csoportok). Egy ismeretlen ok miatt az alkalmazottak új helyre kerültek az épületen belül. Az új felállást és a hálózathoz való csatlakozási információkat az ÁBRA szemlélteti.



ábra 4.28 - Az új felállást szemléltető ábra. Az adott területen dolgozók ugyanabban a VLAN-ban vannak, elkülönítve a többi osztály dolgozóitól.

A fenti ábra tartalmára összegyűjtöttem a csatlakozási információkat a következő táblázatban (táblázat 4.3). Itt egyértelműen eldönthető, hogy melyik eszköz, eddig hol helyezkedett el a hálózatban, illetve mi az új pozíciója és melyik VLAN-hoz van rendelve. A táblázatot osztályonként tagoltam a könnyebb átláthatóság érdekében.

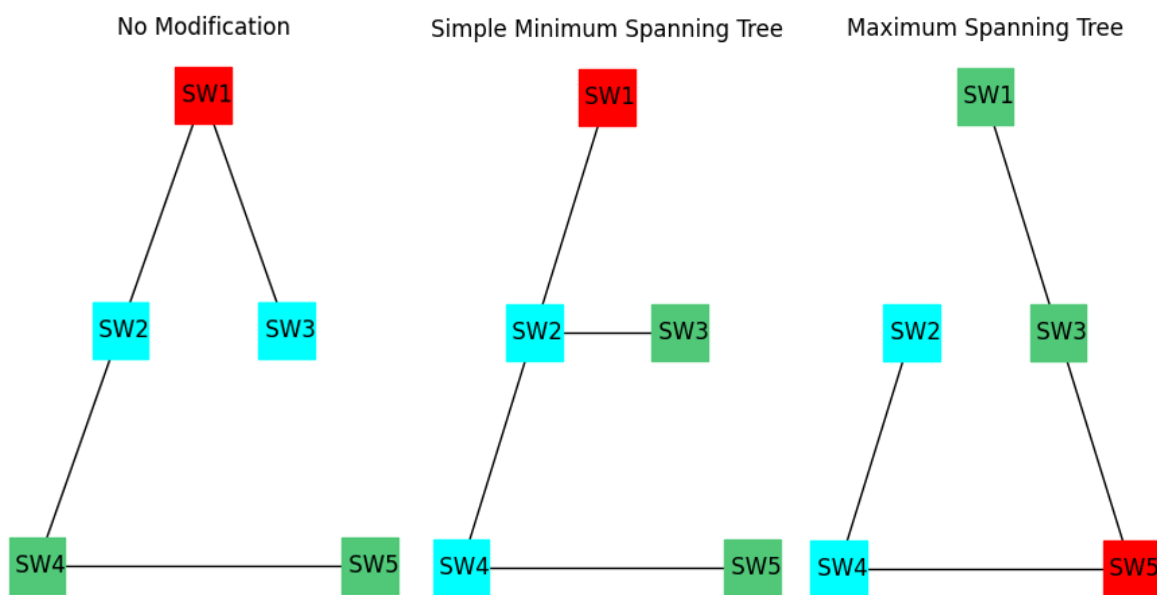
Eszköz	Átrendeződés előtti csatlakozás	Átrendeződés utáni csatlakozás
HR-osztály - VLAN 100		
HR1	SW1	SW1
HR2	SW4	SW3
HR3	SW5	SW5

Sales-osztály - VLAN 200		
SALES1	SW1	SW2
SALES2	SW5	SW3
IT-osztály - VLAN 300		
IT1	SW2	SW1
IT2	SW3	SW3
IT3	SW5	SW4

táblázat 4.3 - Csatlakozási információk az egyes VLAN-ok szerint.

Az átrendeződés után a program futtatási eredményeit a következő ábrák igyekeznek átadni az egyes VLAN-okra bontva. Az első példaesethez képest hasonló formátumban és sorrendben. Eltérés, hogy a hálózati topológia kirajzolása nem szükséges, mivel minden VLAN-ra ugyanaz. Ezért csak az alapfeszítőfa, az egyszerű minimális feszítőfa és a szakdolgozati program eredménye lesz látható.

VLAN 100

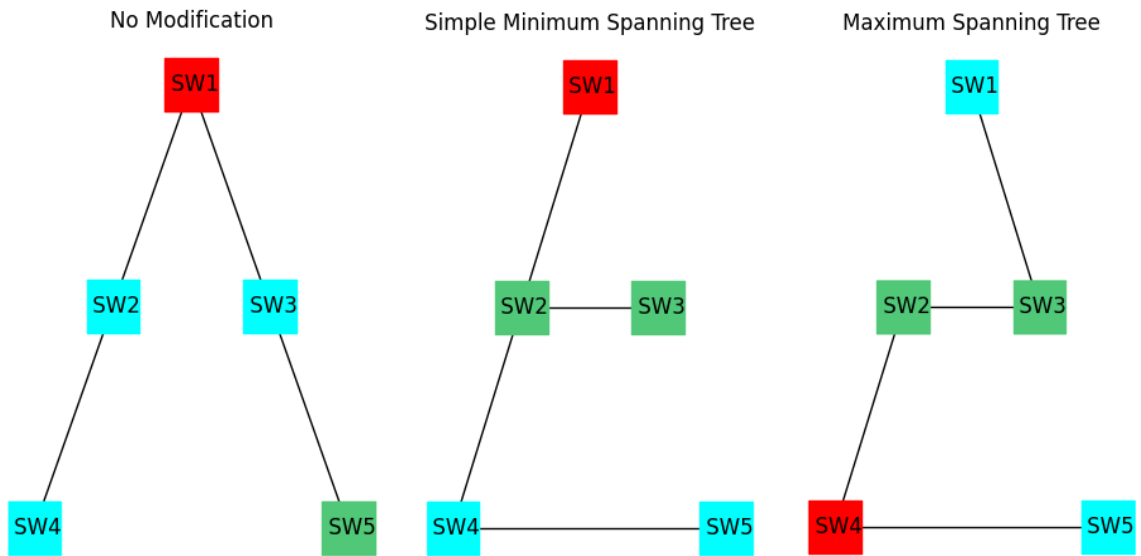


ábra 4.29 - Alapfelállítás feszítőfája -
VLAN 100

ábra 4.30 - Egyszerű minimális
súlyú feszítőfa - VLAN 100

ábra 4.31 - Saját maximális
súlyú feszítőfa - VLAN 100

VLAN 200

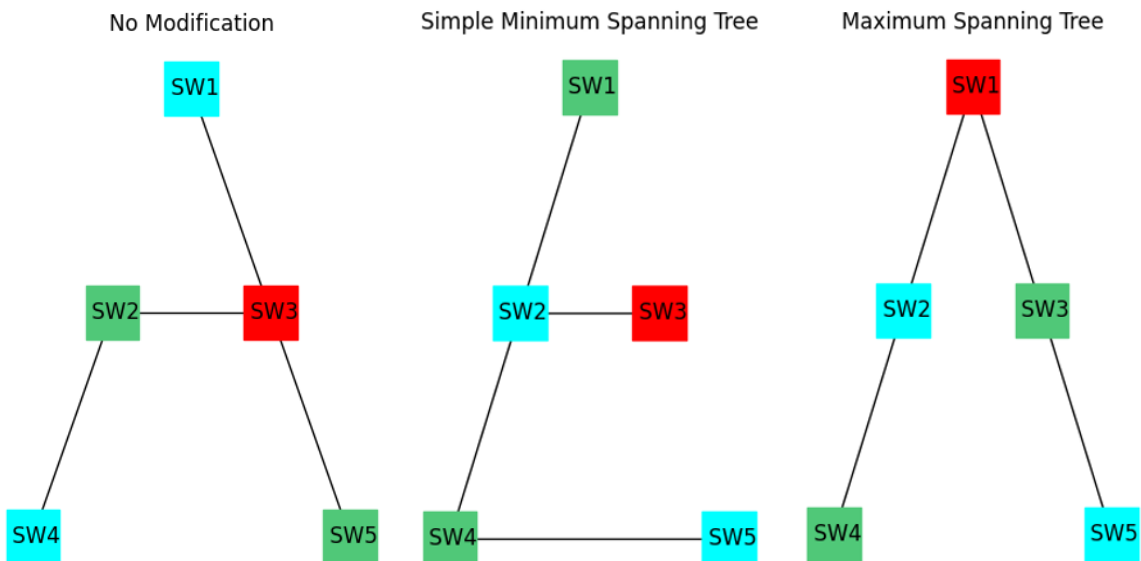


ábra 4.32 - Alapfelállítás feszítőfája -
VLAN 200

ábra 4.33 - Egyszerű minimális
súlyú feszítőfa - VLAN 200

ábra 4.34 - Saját maximális súlyú
feszítőfa - VLAN 200

VLAN 300



ábra 4.35 - Alapfelállítás feszítőfája -
VLAN 300

ábra 4.36 - Egyszerű minimális
súlyú feszítőfa - VLAN 300

ábra 4.37 - Saját maximális súlyú
feszítőfa - VLAN 300

A fenti eredményekből látszódik, hogy egy hálózati feszítőfa sokféleképpen kialakulhat, főleg összetettebb topológiák során. Az viszont nem bizonyos, hogy az egyes kialakult fák megfelelően ellátják a forgalom igényeit és az sem biztos, hogy nem létezik az aktuális kialakításnál egy jobb megoldás.

5 Kiértékelés

Az előző fejezetben kapott eredményeket szeretném kiértékelni ebben a részben, kettő fő szempont alapján. Az első szempont a feszítőfákban kialakult átlagos távolság összehasonlítása a különböző módszerek és az én programom eredménye között. A második szempont pedig az STP konvergenciaidő bemutatása a program futása során, vagyis hogy a hálózat mennyi ideig nem továbbít adatszegmenseket a switch-ek között.

A mért és számolt eredményeket igyekszem jól átlátható rendszerben szemléltetni a könnyebb megértés érdekében.

5.1 Átlagos távolságok vizsgálata

Ahogy azt a 3.1 számú fejezetben már bevezettem, az átlagos távolság meghatározásához szükséges minden access switch (zöld) húzódó távolságok középértékének kiszámolása, ahol a szomszédos switch-ek közötti távolság az egységnyi.

Ez a kiértékelési szempont azt mutatja meg számunkra, hogy mennyire sok kapcsolat szükséges az eszközök között. Ha ez a szám nagy, akkor sok felesleges linken haladnak keresztül a szegmensek. Ha alacsonyabb, akkor pedig minimális a feleslegesen igénybe vett kapcsolatok száma. Ez load balancing (terheléelosztás) szempontból fontos kérdés lehet főleg, ha több tíz - vagy száz VLAN létezik a hálózaton. A különböző fa kialakítások csökkenteni képesek a terheltséget, illetve az esetleges adatütközéseket is.

Az előző fejezet megoldásai közül a második eset eredményeit szeretném alapul venni a távolságok kiszámolásához. Minden egyes VLAN-ra lefuttatott eredmény több különböző kialakítással összehasonlítható, így látható az általam meghatározott módszer hatékonysága, valamint hogy az adott esetben megérte-e az átalakítás.

A Root Bridge helyzete a topológiában fontos szempont az esetleges linkhibák és véletlen átrendeződések hirdetése miatt. A legjobb egy központi hely a fában, úgyhogy ezt a kiértékelésnél is figyelembe szeretném venni, ha az alapfelálláshoz képest felesleges root módosítás történt.

VLAN 100 - HR			
Access Switch:		SW1, SW3, SW5	
Eredmény	Ábra	Átlagos távolság	Root változtatás szükséges? ²
1. Alap feszítőfa.	ábra 4.29	$\frac{1 + 3 + 4}{3} = \frac{8}{3}$	nem
2. Minimális feszítőfa.	ábra 4.30	$\frac{2 + 3 + 3}{3} = \frac{8}{3}$	igen
3. Maximális feszítőfa.	ábra 4.31	$\frac{1 + 2 + 1}{3} = \frac{4}{3}$	igen

táblázat 5.1 - VLAN 100 futtatási eredmények kiértékelési összesítése. A legkisebb átlagos távolságot biztosító feszítőfa kialakítás: 3. Maximális feszítőfa.

A VLAN 100-ba tartozó HR-osztály eszközei között a legkisebb átlagos távolságot az általam készített program biztosította. Ezt az tükrözi, hogy a hálózati változások miatt indokolt volt egy új fa kialakítása optimalizálás gyanánt. A többi esetben a nagyobb az átlagos távolság, vagyis több switch-en halad át az információ. A Root Bridge helyzetét, pedig indokolt volt megváltoztatni az alapfelálláshoz képest.

VLAN 200 - SALES			
Access Switch:		SW2, SW3	
Eredmény	Ábra	Átlagos távolság	Root változtatás szükséges?
1. Alap feszítőfa.	ábra 4.32	$\frac{2}{2} = 1$	nem
2. Minimális feszítőfa.	ábra 4.33	$\frac{1}{2} = 0,5$	igen
3. Maximális feszítőfa.	ábra 4.34	$\frac{1}{2} = 0,5$	igen

táblázat 5.2 - VLAN 200 futtatási eredmények kiértékelési összesítése. A legkisebb átlagos távolságot biztosító feszítőfa kialakítások: 2. Egyszerű minimális feszítőfa és 3. Maximális feszítőfa.

A VLAN 200-hoz tartozó STP beállítások alapján a minimális súlyú feszítőfa algoritmus és az általam készített program is ugyanakkora távolságot reprezentál. Fontos

² A Root Bridge megváltoztatása szükséges-e az alapfelálláshoz képest?

megjegyezni, hogy a minimális súlyú feszítőfa algoritmus a MAC-címek növekvő sorrendjében vizsgálja és alakítja ki a fát, ezért minden VLAN-hoz megegyező alakú fát biztosít. Ebben az esetben éppen annyira ideálisnak bizonyult, mint az én javasolt módszerem.

VLAN 300 - IT			
Access Switch:		SW1, SW3, SW4	
Eredmény	Ábra	Átlagos távolság	Root változtatás szükséges?
1. Alap feszítőfa.	ábra 4.35	$\frac{1 + 3 + 2}{3} = 2$	nem
2. Minimális feszítőfa.	ábra 4.36	$\frac{2 + 2 + 2}{3} = 2$	igen
3. Maximális feszítőfa.	ábra 4.37 ábra 4.31	$\frac{1 + 2 + 3}{3} = 2$	igen

táblázat 5.3 - VLAN 300 futtatási eredmények kiértékelési összesítése. A legkisebb átlagos távolság mindegyik fa felépítésnél megegyezik.

Érdekes eredményt jelöl a táblázat 5.3, hiszen az összes fa átlagos switch-távolsága megegyezik. Ez szépen reprezentálja, hogy kialakulhat olyan eset is, amikor a fa alakjának megváltoztatása szükségtelen. Erre egy továbbfejlesztési lehetőség lehet, ha a módszer először kiszámolja ezeket a távolságokat és csak akkor változtat a fa alakján, ha talál jobb elrendezést. Természetesen egy hatalmas hálózat több switch-én lévő STP kisebb valószínűséggel mutatna azonos távolsági értékeket.

Ami viszont érdekes lehet, az a Root Bridge elhelyezkedése. Alapvető második rétegbeli tervezési szempont, hogy a gyökér kapcsolót a legfelső központi szintre helyezik el (SW1), azonban ekkor a pozíciója állandó. A továbbfejlesztési szempont része lehet a Root Bridge megváltoztatása is. Ugyanis, ha megéri átalakítani a fát, akkor a Root csakis maximális fokszámú csúcs helyén helyezkedhet el az STP tulajdonságai miatt.

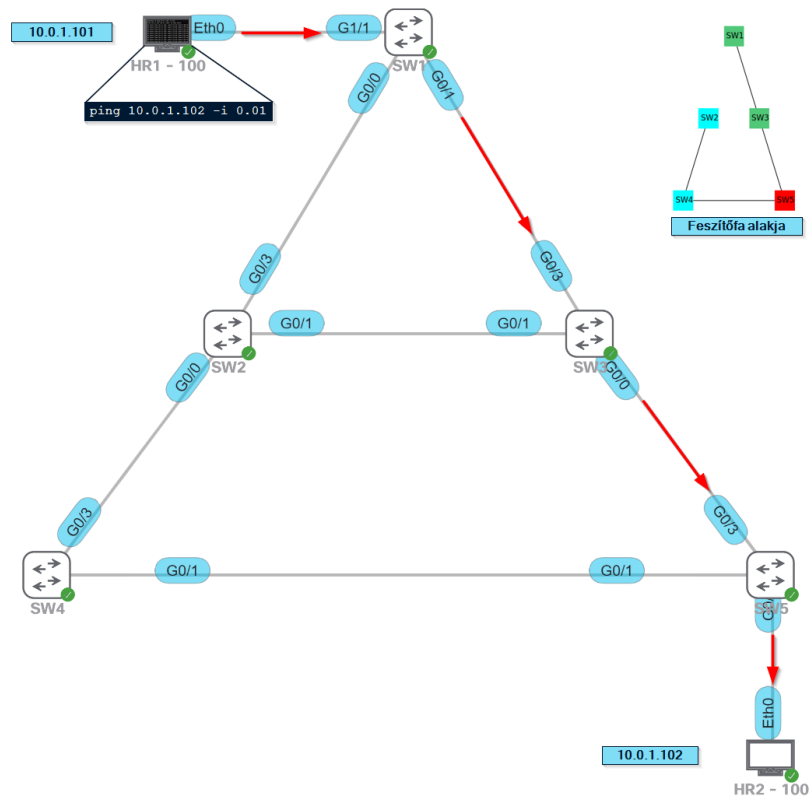
Az STP-ről közismert átrendeződési idő (konvergencia) érdekes vizsgálati szempont lehet a módszer vizsgálata szempontjából. Ha az STP fában változás történik, arról minden eszköznek értesülnie kell. A programozott átalakítás során mindenképpen létrejön valamiféle változás, amit a következő alfejezet taglal.

5.2 Konvergencia idő mérése

Az esetleges linkhibák során (mind fizikai, mind hálózati szinten) a fa újrarendezése és az összes eszköz értesítése a változásról (BPDU üzenetek segítségével), egy bizonyos időt vesz igénybe. Ez idő alatt az adatszegmensek nem kerülnek továbbításra, a végpontok közötti kommunikáció megszakad. Az eredeti STP-hez képest a megoldás során alkalmazott Rapid PVST+, a nevéből is adódóan gyorsabb konvergencia idővel üzemel. Ennek szemléltetésére találtam ki egy mérési módszert.

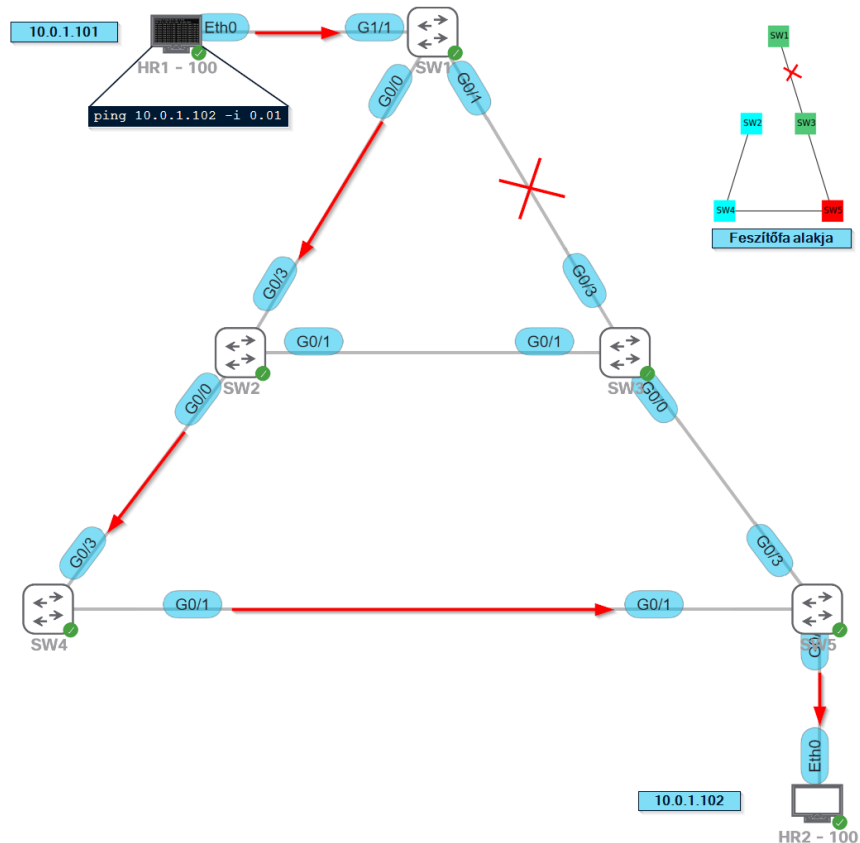
A végfelhasználó eszközökön a *ping* parancs segítségével vizsgálom a kommunikáció során elvesztett szegmensek számát, adott idő alatt. A *ping* parancs paraméterei közé meg lehet adni, a küldött üzenet között eltelt időt is. Hogy minél pontosabb legyen a mérés, a lehető legkisebb időközöt alkalmaztam.

Első körben a Rapid PVST+ konvergencia idejét szeretném megvizsgálni, amit a HR1 számítógépről indított üzenetszekvenciával valósítok meg. A HR1 PC az SW1 számú switch-hez kapcsolódva éri el a hálózatot. A *ping* paranccsal a hálózat ellentétes oldalán található SW5 switch-hez kötött HR2 eszközt célozza, 0,01 másodperces időközökkel. Az adatszegmensek útját az ábra 5.1 szemlélteti, ahol az aktuális feszítőfa is jelölve van.



ábra 5.1 - A *ping* üzenet útja a konvergencia teszt elején.

Folyamatos *ping* üzenetek mellett a tesztelés céljából törlöm SW1 és SW3 közötti linket, így az Rapid PVST+ protokoll új utat kell válasszon, illetve új fát kell kialakítson. A *ping* üzenetek ekkor nem érkeznek meg HR2 PC-hez, ahonnan nem érkezik ICMP válaszüzenet. Amint a BPDU üzenetek segítségével a hálózati fa újra feláll a ping-válaszok újra érkeznek.



ábra 5.2 - A *ping* üzenet új útja, a link törlése után.

Az üzenetek időintervallumából, az elküldött - és az elveszett üzenetek számából ezután egyszerűen meg lehet határozni a keresett konvergencia időt. Mivel a mérési mód nem teljesen pontos ezért több mérés alapján vontam le a várakozási időről következtetéseket. Az eredményeket a (táblázat 5.4) tartalmazza.

A Rapid PVST+ konvergencia ideje után az általam készített program során keletkezett várakozási időt is szeretném megvizsgálni (táblázat 5.5), ugyanezzel a módszerrel. Azonban ebben az esetben nem link hiba generálás az indikátor, hanem az fa-átalakító program futtatása. A Root Bridge helye és a port költségek változása is képes az információtovábbítást valamennyi időre megakasztani, ezért érdemes itt is vizsgálatokat folytatni.

Mérés	Elküldött üzenetek száma [db]	Sikeres üzenetek száma [db]	Veszteség [%]	Konvergencia [mp]
1.	1576	1110	29,57	4,66
2.	1604	1006	37,28	5,98
3.	1380	809	41,38	5,71
4.	1154	613	46,88	5,41
5.	1279	791	38,15	4,88
6.	2117	1586	25,08	5,31
7.	1964	1501	23,57	4,63
Átlagos konvergencia idő:				5,23

táblázat 5.4 - Rapid PVST+ konvergencia idők mérési eredményei. Átlagos konvergencia: 5,23 másodperc. Időintervallum 0,01 másodperc.

Mérés	Elküldött üzenetek száma [db]	Sikeres üzenetek száma [db]	Veszteség [%]	Konvergencia [mp]
1.	2768	2629	5,02	1,39
2.	2419	2361	2,4	0,58
3.	2049	1969	3,9	0,80
4.	2325	2276	2,11	0,49
5.	2122	2033	4,19	0,89
6.	2093	2047	2,2	0,46
7.	2386	2275	4,65	1,11
Átlagos konvergencia idő:				0,82

táblázat 5.5 - Szakdolgozat program működése közben fellépő hálózati konvergencia idő. Átlagos konvergencia: 0,82 másodperc. Időintervallum 0,01 másodperc.

Ahogy az a fenti eredményekből is szépen látszik, a link hibához képest a program konvergencia ideje jóval kevesebb. Ez azt jelenti, hogy az STP beállítási paramétereket automatizáció segítségével való átírása kevesebb ideig blokkolja a hálózat forgalmát, mint az alapbeállításokkal működő STP. Ez azzal indokolható, hogy a BPDU üzeneteknek van egy elévülési idejük az egyes eszközökben. Alapbeállításokkal a BPDU üzenetek kettő másodpercenként kerülnek továbbküldésre (*helloTime* - két BPDU között eltelt idő beállítására szolgál).

Természetesen a beállítások finomhangolásával csökkenthető ez az idő. A Rapid PVST+ esetében is elérhető néhány tized másodperces konvergencia eredmény. Az elkészült program tekintetében pedig egy továbbfejlesztési lehetőség lehet, ha nem a teljes fa kerül módosításra, hanem csak az egyes részei. Például, egyetlen él helyének megváltoztatása kevesebb konvergencia időt eredményez, mint az összes él és a Root helyének cseréje.

5.3 Konklúzió

Az STP protokoll több évtizede megelőzi a kört tartalmazó gráfok kialakulását, ezzel elősegítve a hibátlan L2 szintű adattovábbítást. A technológia még manapság is megállja a helyét több nagyvállalati hálózatban, ami az alkalmazottak és végpontok számának növekedésével bonyolódhat. Főleg, ha az alkalmazott hálózat folytonos változás alatt áll.

A szakdolgozati feladatomban erre a problémára kerestem egy módszert, ami a dinamikus változásokat képes lekövetni és a kialakított feszítőfák alakját karbantartani. A kidolgozott módszer egyféle megközelítés a sok közül, ami a rendelkezésre álló eszközök, rendszerek és a szükséges háttértudás segítségével megoldható volt.

Az előzőekben bemutatott kiértékelési eredmények alapján elmondható, hogy a hálózati feszítőfák átalakítására készült program sikeresen végrehajtotta a feladatot. A példa esetek alapján is képes volt egy optimális fa kialakítást eredményre juttatni. Természetesen vannak olyan esetek, amikor egyéb egyszerűbb algoritmusok is azonos eredményt produkáltak, azonban jobbat soha. A megemlíttet továbbfejlesztési lehetőségek is azt mutatják, hogy van még fejlődési lehetőség, ami ezen projekt keretei közé nem fért be.

A konvergencia idő alapú mérések pedig jól szemléltetik a program effektív futását, ami továbbfejlesztésekkel akár radikálisan csökkenthető. A konvergencia mérés mellett érdemes lehet az átviteli sebességet is mérések alá vetni főleg, ha a rendszer alkalmazása egy olyan nagyvállalatnál valósulna meg, ahol a sebesség mindennél fontosabb.

Automatizáló technológiának a NETCONF protokollt választottam az eddigi gyakorlatom és a könnyű kezelhetősége miatt, azonban egyéb megoldásokkal is megvalósítható ugyanez az eredmény. A RESTCONF protokoll például szinte azonos módon tud működni, csak SSH kapcsolódás helyett http-n keresztül képes az eszközök beállításait módosítani. Ezzel a megoldással például a világ különböző pontjain lévő Enterprise hálózatokat képes egy központi kontroller módosítani.

Elmondható tehát, hogy az STP átkonfigurálásával kialakítható egy olyan módszer, ami központilag optimalizálja a feszítőfa alakját. Mivel minden egyes hálózati rendszer más, ezért érdemes az adott hálózatra fókuszált megoldást megvalósítani, azonban a szakdolgozatomban leírt technológiák és könyvtárak hálózattól függetlenül is alkalmazhatóak.

Irodalomjegyzék

- [1] International Organization for Standardization (ISO): *35.100 - OPEN SYSTEMS INTERCONNECTION (OSI)*, <https://www.iso.org/ics/35.100/x/>
- [2] Edgeworth B., Rios R. G., Hucaby D., Gooley J. (2020): *CCNP and CCIE Enterprise Core ENCOR 350-401 Official Cert Guide*, ISBN 1-58714-523-5, Cisco Press, pp. 597
- [3] Cisco 2020: *Cisco Data Center Spine-and-Leaf Architecture: Design Overview White Paper*, <https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/white-paper-c11-737022.html> (2021. dec.)
- [4] Edgeworth B., Rios R. G., Hucaby D., Gooley J. (2020): *CCNP and CCIE Enterprise Core ENCOR 350-401 Official Cert Guide*, ISBN 1-58714-523-5, Cisco Press, pp. 819-820
- [5] IETF: *Network Configuration Protocol (NETCONF) RFC 6241*, <https://datatracker.ietf.org/doc/html/rfc6241>
- [6] IETF: *RESTCONF Protocol RFC 8040*, <https://datatracker.ietf.org/doc/html/rfc8040>
- [7] IEEE Standards Association: *IEEE 802.1D-1990 - Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*, https://standards.ieee.org/standard/802_1D-1990.html
- [8] IEEE Standards Association: *ISO/IEC 15802-3:1998 802.1w - Rapid Reconfiguration of Spanning Tree* <https://www.ieee802.org/1/pages/802.1w.html>
- [9] IEEE Standards Association (2006): *802.1s - Multiple Spanning Trees*, <https://www.ieee802.org/1/pages/802.1s.html>
- [10] Network Analysis in Python: *Maximum spanning tree algorithm*, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.tree.mst.maximum_spanning_tree.html
- [11] Katona Gy., Recski A., Szabó Cs. (2002): *A számítástudomány alapjai*, Typotex, Budapest, pp. 27
- [12] Edgeworth B., Rios R. G., Hucaby D., Gooley J. (2020): *CCNP and CCIE Enterprise Core ENCOR 350-401 Official Cert Guide*, ISBN 1-58714-523-5, Cisco Press, pp. 42.
- [13] Edgeworth B., Rios R. G., Hucaby D., Gooley J. (2020): *CCNP and CCIE Enterprise Core ENCOR 350-401 Official Cert Guide*, ISBN 1-58714-523-5, Cisco Press, pp. 39.

- [14] Internet Engineering Task Force (IETF) Standards (2011. jún.): *Network Configuration Protocol (NETCONF)* - rfc6241, <https://datatracker.ietf.org/doc/html/rfc6241> , (2021. dec.)
- [15] Internet Engineering Task Force (IETF) Standards (2003. máj.): *Overview of the 2002 IAB Network Management Workshop* - rfc3535, <https://datatracker.ietf.org/doc/html/rfc3535>, (2021. dec.)
- [16] Yu, J; Ajarmeh, I, A 2010, ‘An Empirical Study of the NETCONF Protocol’, *2010 Sixth International Conference on Networking and Services*, IEEE, Cancun, Mexico, pp. 256.
- [17] Python Package Index (PyPi): *ncclient 0.6.12* (2021. máj.), <https://pypi.org/project/ncclient/>
- [18] Internet Engineering Task Force (IETF) Standards (2010. okt.), *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, <https://datatracker.ietf.org/doc/html/rfc6020> , (2021. nov.)
- [19] YANG Central (2016.): *A quick YANG example*, <http://www.yang-central.org/twiki/bin/view/Main/WebHome>, (2021. dec.)
- [20] NetworkX - Network Analysis in Python: *Software for complex networks*, <https://networkx.org/>
- [21] Matplotlib: *matplotlib.pyplot*, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html
- [22] Networkx- Network Analysis in Python: *Maximum Spanning Tree*, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.tree.mst.maximum_spanning_tree.html
- [23] Networkx- Network Analysis in Python: *Shortest Path*, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.shortest_path.html#networkx.algorithms.shortest_paths.generic.shortest_path
- [24] Networkx- Network Analysis in Python: *All Simple Paths*, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html#networkx.algorithms.simple_paths.all_simple_paths
- [25] Ncclient: *Ncclient Documentation*, <https://ncclient.readthedocs.io/en/latest/>
- [26] Internet Engineering Task Force (IETF) Standards (2011 jún.): *Network Configuration Protocol (NETCONF)* - rfc 6241, <https://datatracker.ietf.org/doc/html/rfc6241#page-37>, pp. 37.
- [27] Python.org: *Threading - Thread-based parallelism*, <https://docs.python.org/3/library/threading.html>

Függelék

- GitHub tároló elérési útja az elkészült program forrásfájljaihoz:
<https://github.com/sznistvan/thesis-stp-auto>