МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Мобільний додаток для налаштування технології MPLS»

Завідувач
випускаючої кафедри **Довбиш А.С.**
Керівник роботи **Великодний Д.В.**
Студент гр. ІН-61 **Лиховид В.І.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІКТ

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-61 спеціальності “Інформатика” денної форми навчання Лиховида Владислава Ігоровича.

Тема: “ **Мобільний додаток для налаштування технології MPLS**”

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки:1)огляд існуючих рішень; 2)постановка завдання й формування завдань дослідження; 3)Налаштування технології MPLS; 4) програмна реалізація та її опис; 5)висновки.

Дата видачі завдання “_____”_____2020 р.

Керівник випускної роботи_____ Великодний Д.В.

Завдання прийняв до виконання _____ Лиховид В.І.

РЕФЕРАТ

Записка: 49 стор., 20 рис., 4 додатки, 11 джерел

Об’єкт дослідження — Технологія MPLS

Мета роботи — Розроблення мобільного додатку для налаштування технології MPLS

Методи дослідження — Моделювання схеми з налаштуванням MPLS у емуляторі GNS3. Застосування мови програмування Swift для розробки мобільного додатку

Результати — Розроблено мобільний додаток в якому можна ввести дані: ip-адреси та маски для кожного інтерфейсу та отримати налаштування. Через буфер обміну їх можна скопіювати та використовувати для налаштування схеми.

MPLS, CISCO, GNS3, IPHONE, SWIFT

ЗМІСТ

ВСТУП

На сьогоднішній день однією з перспективних транспортних технологій мереж операторів є MPLS. MPLS (Multiprotocol Label Switching) - це мультипротокольна комутація по мітках. Для її налаштування необхідно виконати деяку кількість дій:

- Аналіз технології MPLS
- Побудувати працюючу мережу
- Налаштувати MPLS для кожного роутера

До того ж якщо раніше головними вимогами, що пред'являються до технології магістральної мережі, були висока пропускна здатність, незначне значення затримки і хороша масштабованість, то сьогодні постачальнику послуг вже недостатньо просто надавати доступ до своєї IP-магістралі. Потреби просунутих користувачів вже припускають і доступ до інтегрованих сервісів мережі, і організацію віртуальних приватних мереж (VPN), і безліч інших спеціальних послуг.

Зростаючий попит на додаткові послуги, що реалізуються поверх простого IP-доступу, обіцяє принести Інтернет-провайдерам значні прибутки, що змушує розробників шукати нові шляхи сполучення мереж на основі IP і ATM.

Для вирішення виникаючих завдань компанією Cisco була розроблена нова технологія під назвою Tag Switching, яка трансформувалася в стандарт MPLS (Multiprotocol Label Switching). При розробці технології деякі вдалі ідеї були запозичені у конкуруючої технології IP-комутації компанії Ipsilon і проекту ARIS корпорації IBM. Можна з упевненістю сказати, що MPLS поєднує в собі кращі елементи всіх згаданих розробок.

Технологія MPLS повинна «підніматися» на вже існуючій і нормально функціонуючої IP мережі. Для цього необхідно розібрати технології маршрутизації для швидкого налаштування мережі.

Для вирішення проблеми скорочення часу налаштування необхідно розробити графічний інтерфейс який дозволяв би користувачу швидко отримати налаштування для потрібних даних.

У сучасному світі важко собі уявити життя без смартфону. Його має майже кожен житель розвинутих країн. На смартфоні, за допомогою додатку, користувач зможе швидко налаштувати під себе дану технологію. На сьогодні найпопулярнішим смартфоном - є «IPhone».

Тому вирішенням проблеми буде створити мобільний додаток для IPhone.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Принцип роботи MPLS

Вхідний маршрутизатор з MPLS (multiprotocol label switching) буде позначати пакети даних при вході в мережу розставляючи мітки, тому, маршрутизатори будуть точно розуміти, куди направляються дані, без необхідності знову і знову аналізувати пакет з даними [1].

Щоб зрозуміти принцип роботи методики MPLS слід зазначити, що в традиційній IP-мережі кожному маршрутизатора доводиться виконувати пошук IP, шляхом постійного пошуку його в таблицях з пакетами даних з метою пересилки на наступний рівень поки пакети даних не досягнуть потрібного пункту призначення.

MPLS технологія привласнює мітку всім IP-пакетів, а тим часом вже самі маршрутизатори приймають рішення про передачу пакету далі на наступний пристрій завдяки потрібного значення мітки. Мітка додається в складі MPLS заголовка, який додається між заголовком кадру (другий рівень OSI) і заголовком пакету (третій рівень OSI) і, по суті, в подальшому йде їх накладення один на одного.

Методика MPLS замість цього виконує "комутацію міток", коли перший пристрій виконує пошук маршрутизації, як і раніше, але замість пошуку наступного переходу він знаходить кінцевий маршрутизатор призначення по заздалегідь заданому маршруту. Маршрутизатор визначає мітку на основі інформації, яку будуть використовувати маршрутизатори для подальшої маршрутизації трафіку без необхідності будь-яких додаткових пошуків IP адрес, по досягненню кінцевого маршрутизатора мітка видаляється і пакет доставляється за допомогою звичайної IP маршрутизацією.

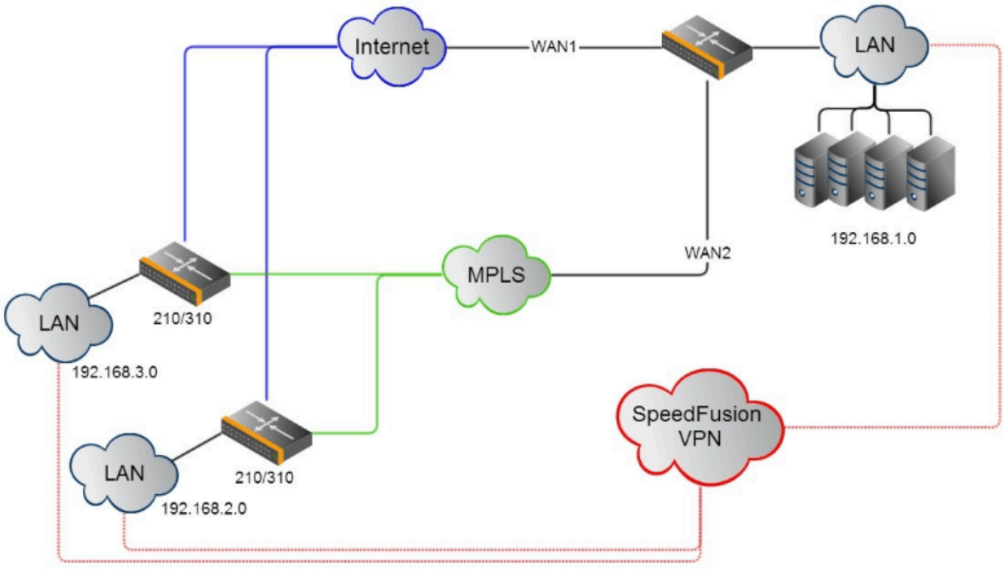

Рисунок 1.1 — Схема з прикладом використання MPLS в мережі

Джерело: Протоколи і стандарти. MPLS як працює на навіщо потрібен [Електронний ресурс] - *https://wiki.merionet.ru/seti/25/mpls-kak-rabotaet-i-zachem-nuzhen/*

- Система міток значно знижує час, необхідний на пошук IP-маршрутизації.
- Дозволяє здійснювати точний пошук збігів з найдовшим префіксом, що знижує ресурс звернення до пам'яті для маршрутизації одного пакета.
- Точні збігу на основі міток набагато простіше реалізувати в обладнанні при меншому навантаженні на нього.
- Дає можливість контролювати, де і як трафік розподілений в мережі, щоб управляти пропускною спроможністю, розставляти пріоритети для різних сервісів і запобігати перевантаження устаткування.

Для роботи MPLS використовують протоколи маршрутизації розповсюдження влучний (LDP), простий необмежений протокол (без підтримки трафіку), протокол резервування ресурсів з проектуванням

трафіку (RSVP-TE). На практиці ж зазвичай використовують протокол розповсюдження влучний (LDP), однак протокол RSVP-TE необхідний для функцій організації трафіку і в складних мережах практично не обійтися без цих двох протоколів з налаштуванням LDP для тунелювання всередині протоколу RSVP.

Передача і управління трафіків відбувається за рахунок технології Traffic Engineering, яка здійснює передачу трафіку по каналах по найбільш оптимальним маршрутом, але з деякими обмеженнями завдяки технології CSPF (Constrained Shortest Path First), яка вибирає шляхи не тільки користуючись критерієм, заснованому на його оптимальній довжині маршруту , але ще і враховує завантаження маршрутів. Використовувані протоколи RSVP-TE дозволяють резервувати смуги пропускання в мережі.

Технологія MPLS також має захист від збоїв грунтуючись попередньому розрахунку шляхів резервного копіювання для потенційних збоїв каналу або вузла. При наявності збою в мережі автоматично відбувається розрахунок найкращого шляху, але при наявності одного збою розрахунок необхідної шляху починає відбуватися ще до виявлення збою. Шляхи резервного копіювання попередньо запрограмовані в FIB маршрутизатора в очікуванні активації, яка може статися в мілісекундах після виявлення збою.

1.2 Графічні симулятори для налаштування мереж

Для розуміння технологій налаштування мереж існують симулятори які дозволяють робити працездатні моделі мережі, взаємодіяти з роутерами, комутаторами, та іншими компонентами необхідними для налаштування якісної мережі. Детальніше розберемо найпопулярніші симулятори:

Cisco Packet Tracer - це багатофункціональна програма моделювання мереж, яка дозволяє студентам експериментувати з поведінкою мережі і оцінювати можливі сценарії. Будучи невід'ємною частиною комплексної

середовища навчання мережевої академії, Packet Tracer надає функції моделювання, візуалізації, авторської розробки, атестації та спільної співпраці, а також полегшує викладання і вивчення складних технологічних принципів [2].

Packet Tracer доповнює фізичне обладнання класу, дозволяючи студентам створювати мережі з практично необмеженою кількістю пристроїв, підтримуючи накопичення практичного досвіду, прагнення до відкриттів і розвиток навичок щодо усунення несправностей. Packet Tracer доповнює програми мережевих академій, що дозволяє викладачам легко описати і показати складні технічні принципи і проекти мережевих систем.

Graphical Network Simulator. Якщо перекласти дослівно - графічний симулятор мережі. Він дозволяє створювати різні мережеві топології прямо на вашому комп'ютері. Найчастіше GNS використовується в якості лабораторного стенд, де можна перевірити ту чи іншу технологію або схему [3].

Насправді GNS3 НЕ симулятор, а емулятор! Варто розуміти різницю між цими поняттями.

Емулятор дозволяє створити модель комп'ютера або іншого пристрою і запускати всередині оригінальне програмне забезпечення. Емулюються всі основні компоненти пристрою, в тому числі процесор, пам'ять і пристрої введення / виводу. У випадку з Cisco, емулятор створює модель маршрутизатора і запускає всередині реальну операційну систему Cisco IOS. Таким чином ми отримуємо повнофункціональний маршрутизатор.

Симулятор ж імітує поведінку системи і її інтерфейсу. Яскравий приклад - Cisco Packet Tracer. Програмісти цього ПО просто створили пристрої з схожим інтерфейсом і схожими командами.

Спираючись на вищесказане можна обрати один з цих симуляторів для налаштування схеми MPLS. Мною був обраний симулятор GNS3, так я

він створює модель маршрутизатора і запускає реальну операційну систему Cisco.

1.3 Протоколи динамічної марштутизації

Для налаштування MPLS необхідно спочатку налаштувати мережу використовуючи один із протоколів динамічної маршрутизації.

EIGRP (англ. Enhanced Interior Gateway Routing Protocol) - протокол маршрутизації, розроблений фірмою Cisco на основі протоколу IGRP тієї ж фірми. Реліз протоколу відбувся в 1994 році. EIGRP використовує механізм DUAL для вибору найбільш короткого маршруту [4].

Більш ранній і практично не використовується нині протокол IGRP був створений як альтернатива протоколу RIP (до того, як був розроблений OSPF). Після появи OSPF, Cisco представила EIGRP - перероблений і поліпшений варіант IGRP, вільний від основного недоліку дистанційно-векторних протоколів - особливих ситуацій з зацикленням маршрутів - завдяки спеціальним алгоритмом поширення інформації про зміни в топології мережі. EIGRP більш простий в реалізації і менш вимогливий до обчислювальних ресурсів маршрутизатора ніж OSPF. Також EIGRP має більш просунутий алгоритм обчислення метрики DUAL (Diffusing Update ALgorithm), який може використовувати 5 різних компонентів для розрахунку:

Bandwidth (Пропускна здатність). Мінімальна пропускна здатність для даного маршруту (а не сума цін (cost) на відміну від OSPF).

Reliability (Надійність). Найгірший показник надійності на всьому шляху маршруту, заснований на keepalive.

Loading (Завантаженість). Найгірший показник завантаженості інтерфейсу на всьому шляху маршруту, заснований на кількості трафіку проходить через інтерфейс і налаштованому на ньому параметрі bandwidth.

За замовчуванням, EIGRP використовує тільки перші два компоненти, так як надійність і завантаженість - динамічні величини, які можуть

змінюватися до декількох разів на секунду. Відповідно, кожна зміна викликає перерахунок метрики для маршрутів і використання процесорної потужності маршрутизатора до 100%. MTU не є динамічною величиною, але не використовується через слабке впливу на метрику маршруту.

Варто зауважити, що ускладнення формули обчислення метрик призводить і до ускладнення розуміння метрики адміністратором. Хоча багато прихильників OSPF і вважають, що, за інших рівних, EIGRP відпрацьовує зміна топології повільніше ніж OSPF, але це скоріше оману, оскільки при малій кількості маршрутизаторів EIGRP відпрацьовує швидше, а при ускладненні схеми дизайн та архітектура протоколу OSPF вимагає більш ретельного впровадження (створення зон і міжзонних відносин), що також уповільнює обмін маршрутами і ускладнює кількість обчислень необхідних для вибору кращого маршруту. В результаті EIGRP працює порівняно однаково, а в деяких найпростіших або навпаки більш складних топологиях навіть швидше ніж інші, існуючі на даний момент, протоколи маршрутизації.

Ще одна перевага протоколу EIGRP в тому, що він здатний виробляти суммарізацію на будь-якому маршрутизаторі на шляху, оскільки є протоколом класу «вектор відстані» (Distance Vector), інформація передається від сусіда до сусіда, де кожен наступний вибирає тільки кращий маршрут, що віддають сусідові.

OSPF (Open Shortest Path First) - дослівно перекладається як «Спершу відкритий короткий шлях» - надійний протокол внутрішньої маршрутизації з урахуванням стану каналів (Interior gateway protocol, IGP). Як правило, даний протокол маршрутизації починає використовуватися тоді, коли протоколу RIP вже не вистачає через ускладнення мережі і необхідності в її легкому масштабування [5].

OSPF найбільш широко використовуваний протокол внутрішньої маршрутизації. Коли йде мова про внутрішньої маршрутизації, то це

означає, що зв'язок між маршрутизаторами встановлюється в одному домені маршрутизації, або в одній автономній системі. Уявіть компанію середнього масштабу з декількома будівлями і різними департаментами, кожне з яких пов'язане з іншим за допомогою каналу зв'язку, які дублюються з метою збільшення надійності. Всі будівлі є частиною однієї автономної системи. Однак при використанні OSPF, з'являється поняття «майданчик», «зона» (Area), яке дозволяє сильніше сегментувати мережу, наприклад, поділ по «зонах» для кожного окремого департаменту.

Спершу, коли протокол тільки запустився на маршрутизаторі, він починає посилати hello-пакети для знаходження сусідів і вибору DR (designated router, призначений маршрутизатор). Ці пакети включають в себе інформацію про сусідів і стан каналів. Наприклад, OSPF може визначити з'єднання типу «точка-точка», і після цього в протоколі дане з'єднання «піднімається», тобто стає активним. Якщо ж це розподілене з'єднання, маршрутизатор чекає вибору DR перед тим як помітити канал активним.

Існує можливість змінити Priority ID для, що дозволить бути впевненим в тому, що DR-му стане найпотужніший і продуктивний маршрутизатор. В іншому випадку, переможе маршрутизатор з найбільшим IP-адресою. Ключова ідея DR і BDR (Backup DR), полягає в тому, що вони є єдиними пристроями, що генерують LSA і вони зобов'язані обмінюватися базами даних стану каналів з іншими маршрутизаторами в підмережі. Таким чином, все не-DR маршрутизатори формують сусідство з DR. Весь сенс подібного дизайну в підтримці масштабованості мережі. Очевидно, що єдиний спосіб переконатися в тому, що всі маршрутизатори оперують однією і тією ж інформацією про стан мережі - синхронізувати БД між ними. В іншому випадку, якби в мережі було 35 маршрутизаторів, і потрібно було б додати ще один пристрій, з'явилася б необхідність у встановленні 35 процесів сусідства. Коли база централізована (тобто існує центральний,

обраний маршрутизатор - DR) даний процес спрощується на кілька порядків.

Обмін базами даних - вкрай важлива частина процесу по встановленню сусідства, після того як маршрутизатори обмінялися hello-пакетами. При відсутності синхронізованих баз даних можуть з'явитися помилки, такі як петлі маршрутизації і т.д. Третя частина встановлення сусідства - обмін LSA. Це поняття буде розібрано в наступній статті, головне, що необхідно знати - нульова зона (Area 0) особлива, і при наявності декількох зон, всі вони повинні бути з'єднані з Area 0. Так само це називається магістральної зоною.

Опираючись на вищесказані факти для налаштування MPLS більше підходить OSPF.

1.4 Постановка задачі

Провівши аналіз літератури можна виділити такі висновки. MPLS – технологія яка необхідна в сучасному світі, але для її налаштування необхідно виконати деякі дії.

Постановку задачі можна сформувати наступним чином:

1. Налаштувати схему мережі з використанням технології MPLS у симуляторі GNS3.
2. Проаналізувати схему, та визначити які дії можна спростити за допомогою мобільного додатку.
3. Створити мобільний додаток використовуючи мову програмування – Swift.

2 НАЛАШТУВАННЯ ТЕХНОЛОГІЇ MPLS

2.1 Налаштування схеми використовуючи протокол динамічної марштрутизації OSPF

У симуляторі GNS3 створимо таку схему:

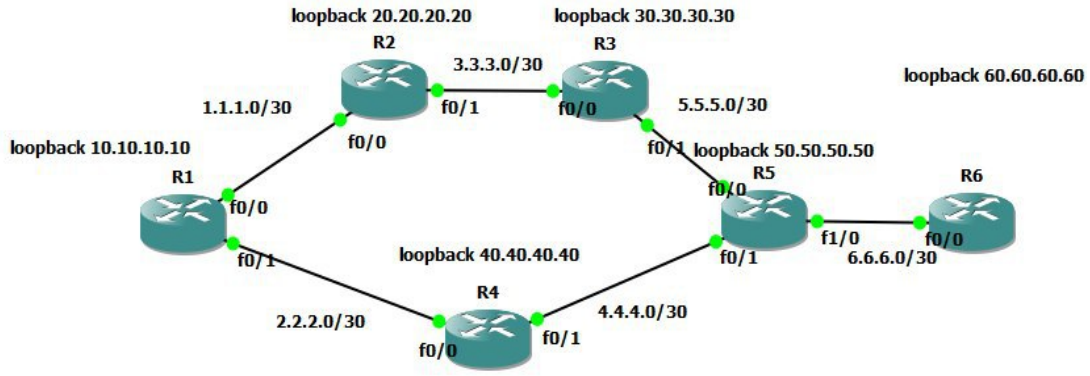

Рисунок 2.1 – Топологія мережі

Для налаштування даної схеми необхідно виконати наступні команди:

Для R1

en

conf t

int loopback 0 – Створюємо loopback інтерфейс

ip address 10.10.10.10 255.255.255.255

exit

int fa 0/0

ip address 1.1.1.1 255.255.255.252

no sh

exit

int fa 0/1

ip address 2.2.2.1 255.255.255.252

no sh

exit

router ospf 1 – Налаштовуємо протокол динамічної маршрутизації OSPF

network 1.1.1.0 0.0.0.3 area 0 –

network 2.2.2.0 0.0.0.3 area 0

network 10.10.10.10 0.0.0.0 area 0

exit

exit

wr

Після налаштувань першого роутера необхідно виконати налаштування для інших роутрерів див. Додаток Д. Після налаштувань необхідно перевірити мережу

Для перевірки мережі з роутера R1 виконаємо команду «ping 60.60.60.60», що відправить пакети на loopback інтерфейс крайнього роутера R6.

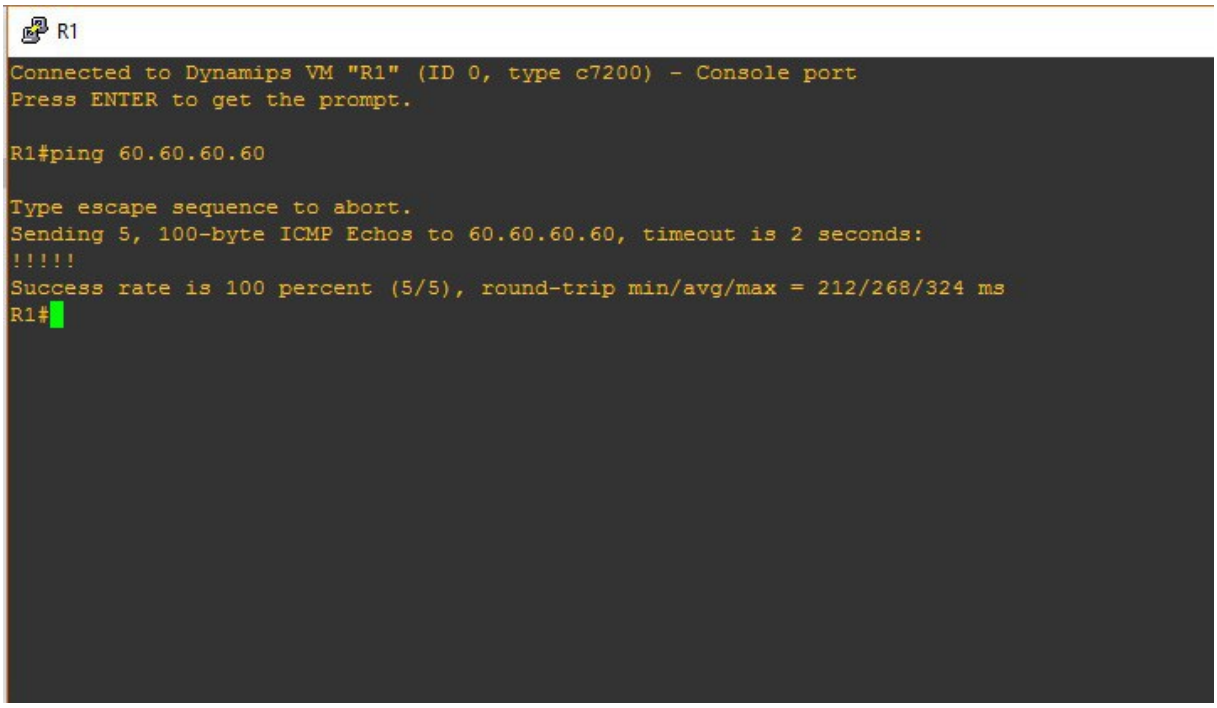

Рисунок 2.2 – Консоль роутера R1

На рис. 2.2 видно «Success rate is 100 percent 5/5», це означає що було послано 5 пакетів і кожен дійшов до адресата 60.60.60.60. Тому можемо зробити висновок що мережа налаштована вірно.

2.2 Налаштування MPLS

Для налаштування MPLS необхідно на кожному роутері виконати наступні команди:

conf t

ip cef. – Вмикаємо Cisco Express Forwarding – технологія швидкої комутації пакетів.

mpls ip – Вмикаємо глобально процес комутації по мітках.

mpls label protocol ldp – Обираємо протокол, по якому будуть обмінюватися мітками LSR (ELSR) між собою (є ще TDP, він є пропрієтарним).

mpls ldp router-id loopback 0 – Визначаємо, який інтерфейс (IP-адреса) береться в якості ID роутера в процесі MPLS.

int fa 0/0

mpls ip – Вмикаємо MPLS на інтерфейсі.

mpls mtu 1512 – Збільшуємо розмір mtu для запобігання фрагментації фреймів.

exit

int fa 0/1

mpls ip

mpls mtu 1512

exit

exit

wr

Для перевірки виконаємо команду sh mpls forwarding-table на роутері R1

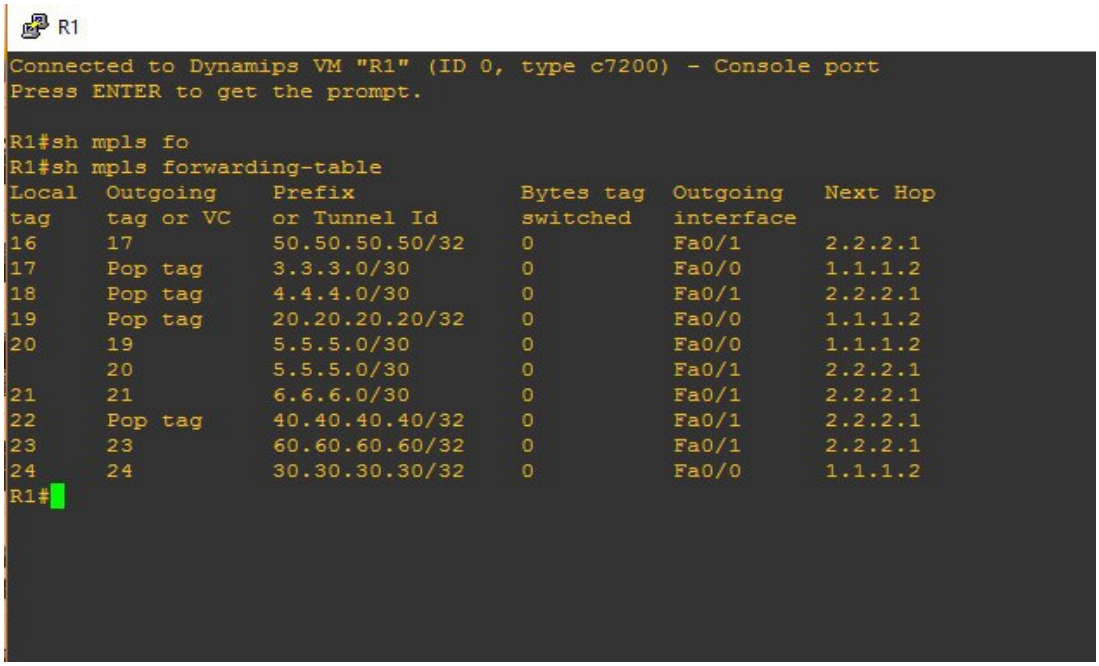

Рисунок 2.3 – Результат команди sh mpls forwarding-table

Як ми бачимо всі маршрути помічені мітками.

Далі виконаємо команду sh mpls ldp neighbor

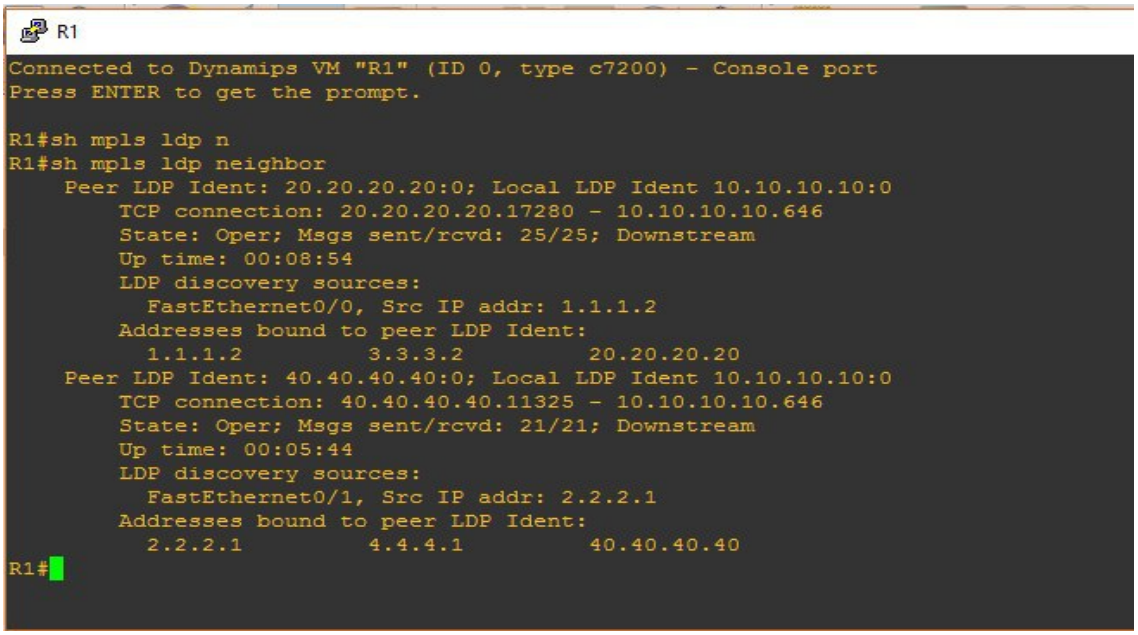

Рисунок 2.4 – Результат команди sh mpls ldp neighbor

Можемо бачити що присутні 2 сусіди, як і повинно бути.

Далі введемо команду sh mpls ip binding

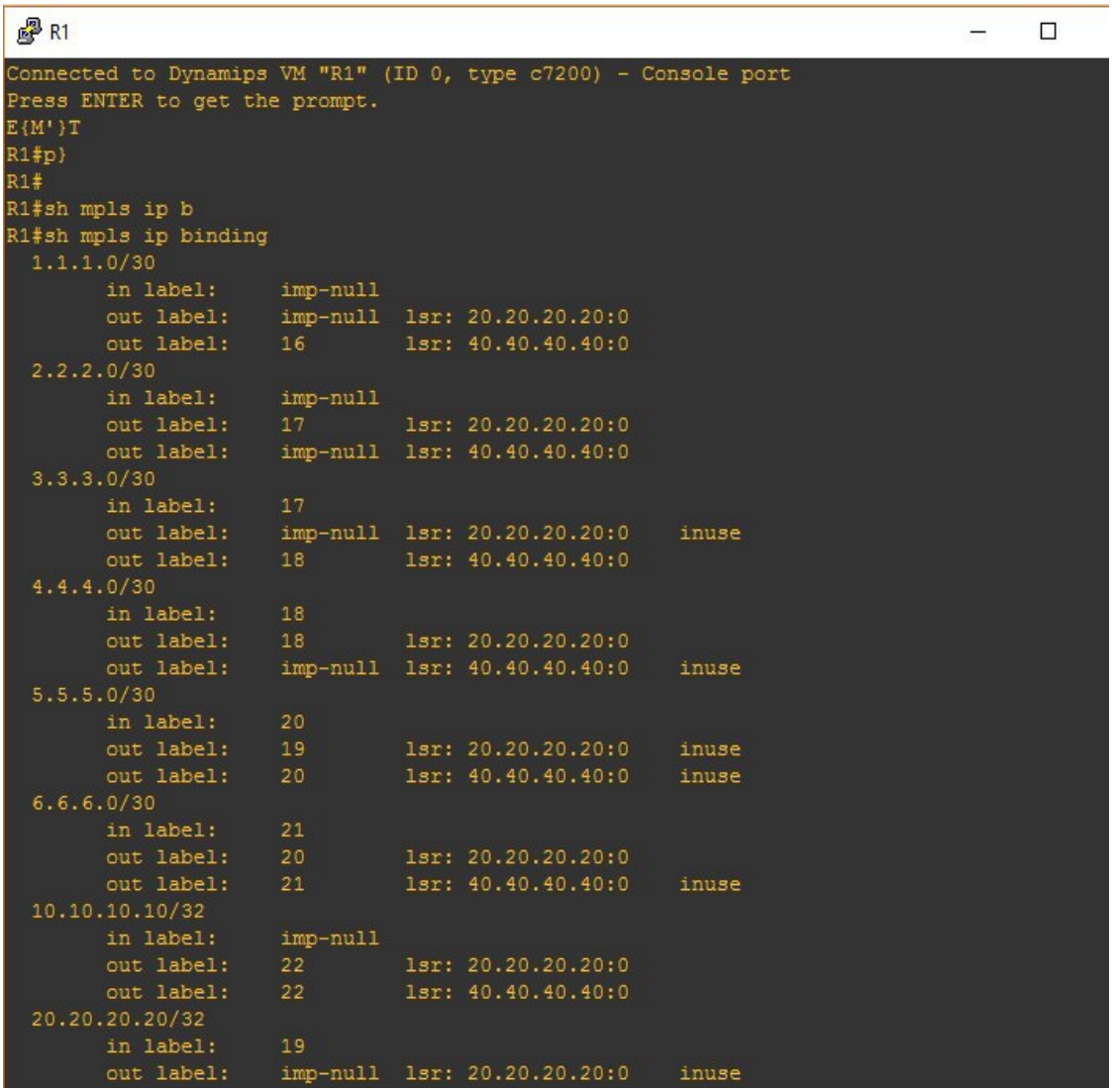

Рисунок – 2.5 Результат команди sh mpls ip binding

Таблиця відповідності міток і маршрутів (з інформацією, який LSR її надав).

Для остаточної перевірки необхідно перевірити ip-пакети за допомогою програми – Wireshark.

Wireshark - програма-аналізатор трафіку для комп'ютерних мереж Ethernet і деяких інших. Має графічний користувальницький інтерфейс. Спочатку проект називався Ethereal, але, через проблеми з торговою маркою, в червні 2006 року проект був перейменований в Wireshark [6].

Функціональність, яку надає Wireshark, дуже схожа з можливостями програми tcpdump, однак Wireshark має графічний користувальницький інтерфейс і набагато більше можливостей із сортування та фільтрації інформації. Програма дозволяє користувачеві переглядати весь проходить по мережі трафік в режимі реального часу, переводячи мережеву карту в нерозбірливий режим (англ. Promiscuous mode).

Програма поширюється під вільною ліцензією GNU GPL і використовує для формування графічного інтерфейсу кроссплатформенную бібліотеку GTK + (планується перехід на Qt). Існують версії для більшості UNIX-подібних систем, в тому числі GNU / Linux, Solaris, FreeBSD, NetBSD, OpenBSD, Mac OS X, а також для Windows.

Wireshark - це програма, яка «знає» структуру самих різних мережевих протоколів, і тому дозволяє розібрати мережевий пакет, відображаючи значення кожного поля протоколу будь-якого рівня. Оскільки для захоплення пакетів використовується pcap, існує можливість захоплення даних тільки з тих мереж, які підтримуються цією бібліотекою. Проте, Wireshark уміє працювати з безліччю форматів вхідних даних, відповідно, можна відкривати файли даних, захоплених іншими програмами, що розширює можливості захоплення. Для розширення можливостей програми можливе використання скриптового мови Lua.

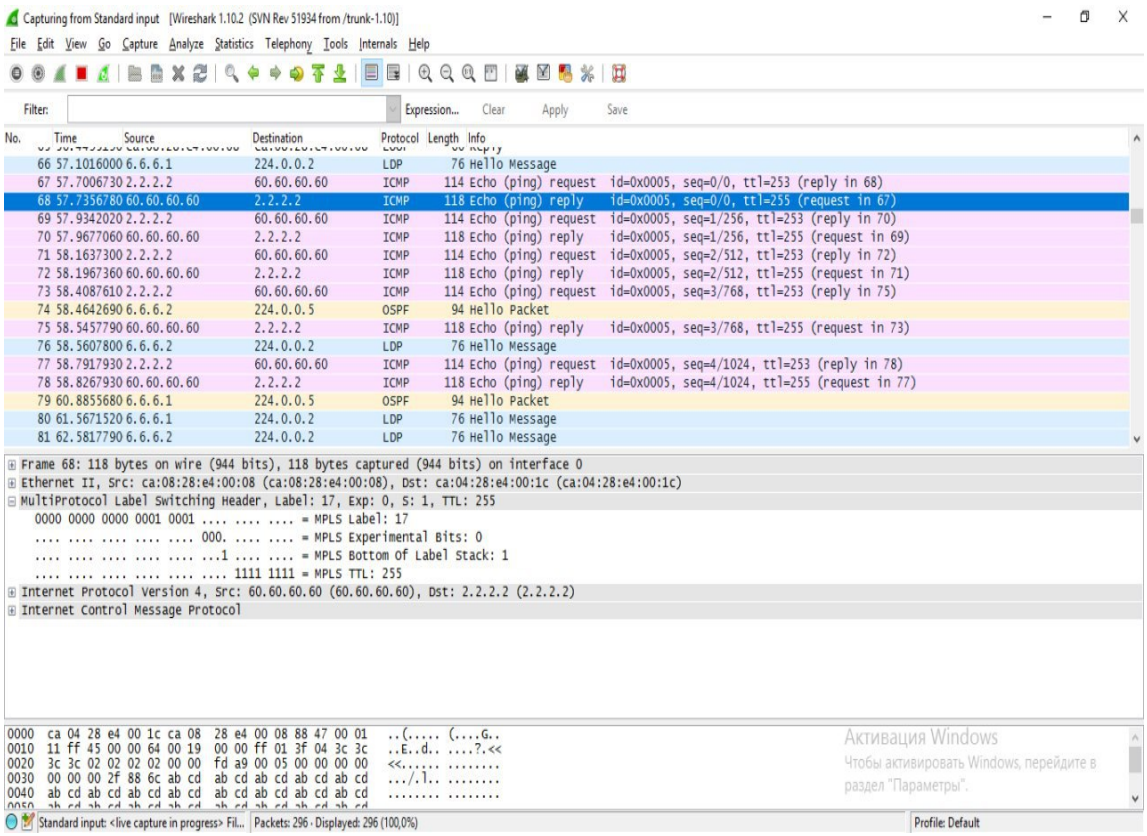

Рисунок 2.6 – Перехоплений ICMP пакет

На рис. 2.6 можемо бачити перехоплений за допомого програми Wireshark ICMP пакет на якому видно MultiProtocol Label Switchigh Header. Що остаточно підтверджує налаштування MPLS.

2.3 Мови програмування додатків для IPhone

Для програмування додатків для IPhone є 2 варіанти мов: Objective-C та Swift.

Objective-C — ("Обджектів Сі") рефлективна, високорівнева об'єктно-орієнтована мова програмування загального призначення, розроблена у вигляді набору розширень стандартної C. Мова програмування Objective-C, розроблена на початку 1980-х років, була основною мовою, що використовувалася компанією NeXT для операційної системи NeXTSTEP, від якої пішли macOS і iOS.На даний час використовується в основному у macOS та GNUStep — середовищах, розроблених на основі стандарту OpenStep, та Cocoa — бібліотеки компонентів для розробки програм.

Програму на Objective-C що не використовує цих бібліотек можна скомпілювати для будь-якої платформи, яку підтримує gcc компілятор з підтримкою Objective-C. Objective-C є розширенням C і тому будь-яку програму на C можна скомпілювати компілятором Objective-C.ООП в Objective-C включає інтерфейси, класи, категорії. Реалізовано одиничне, невіртуальне спадкування. Немає єдиного базового класу для всіх об'єктів. Всі методи в класі — віртуальні. Категорія — парадигма яка дозволяє описувати інтерфейс з методами які «необов'язково» імплементувати [7].

До 2014 року Objective-C була єдина мова програмування для IOS – операційна система IPhone. У 2014 році Apple розробила сучасну мову програмування Swift, яка зараз використовується майже в усіх сучасних додатках. За винятком тих, які були створені на Objective-C і їх перехід на Swift є нераціональним з фінансової, або інших причин.

Swift — багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks [8].

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття

Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування. Основним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням тулкіта Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Сирцевий код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

Swift щільно інтегровано до власницького середовища розробки Xcode, проте може бути викликано з терміналу, що уможливлює її використання на операційний системах, відмінних від macOS, наприклад, на Linux.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Розробка інтерфейсу додатка

На основі законфігурованої мережі у емуляторі GNS3, де були налаштовані роутери з використанням протоколу динмамічної маршрутизації OSPF, та налаштована технологія MPLS. У ході налаштування були виявлені процеси налаштування які можна спростити за допомогою додатку. На основі цього був розроблений інтерфейс додатку.

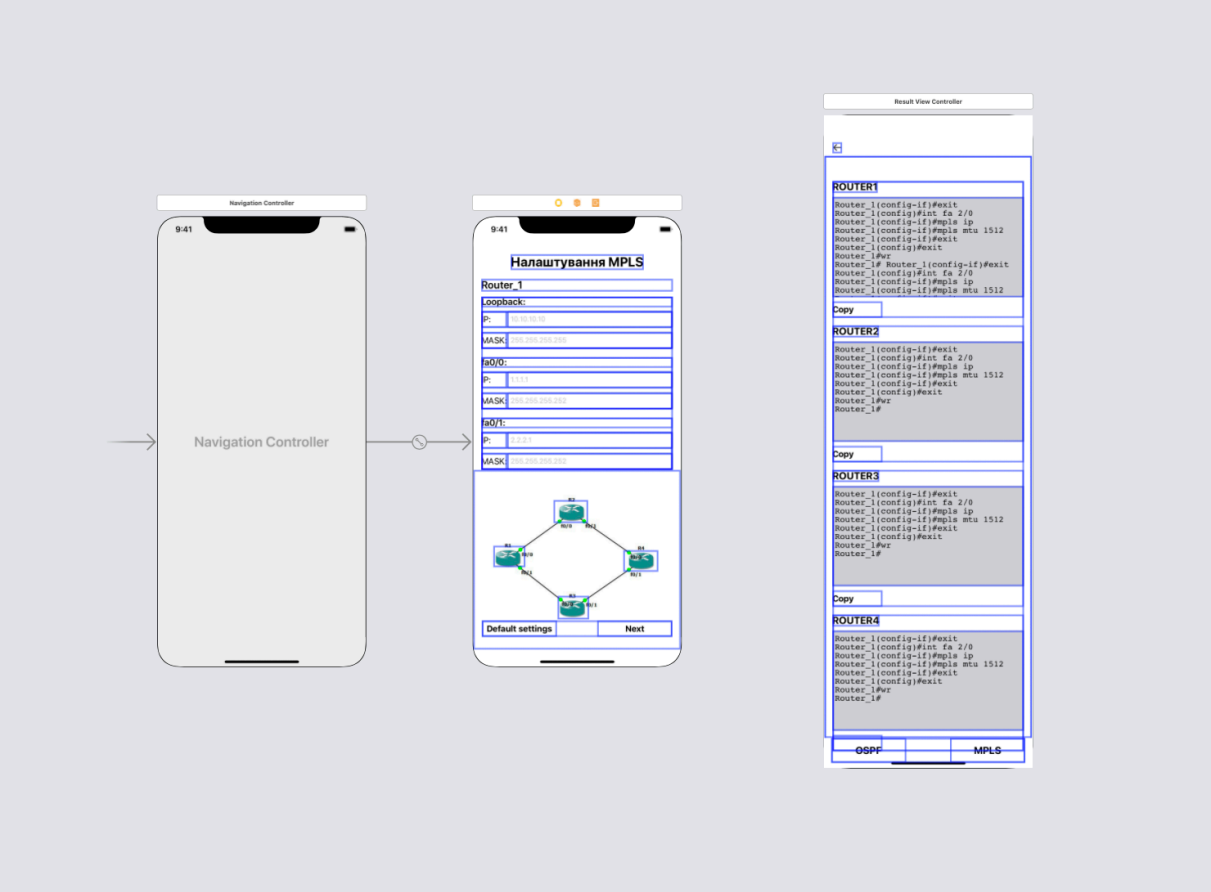

Рисунок 3.1 – Схема екранів

Відкривши додаток, користувач може побачити меню налаштування роутерів.

Рисунок 3.2 – Екран налаштування роутерів

На рис. 3.2 можемо побачити тектові поля для вводу ip-адрес, та масок для кожного інтерфейсу роутера та для loopback. Нижче схему роутерів та кнопки «Default settings» та «Next».

На наступному екрані користувач має можливість побачити команди для налаштування роутерів.

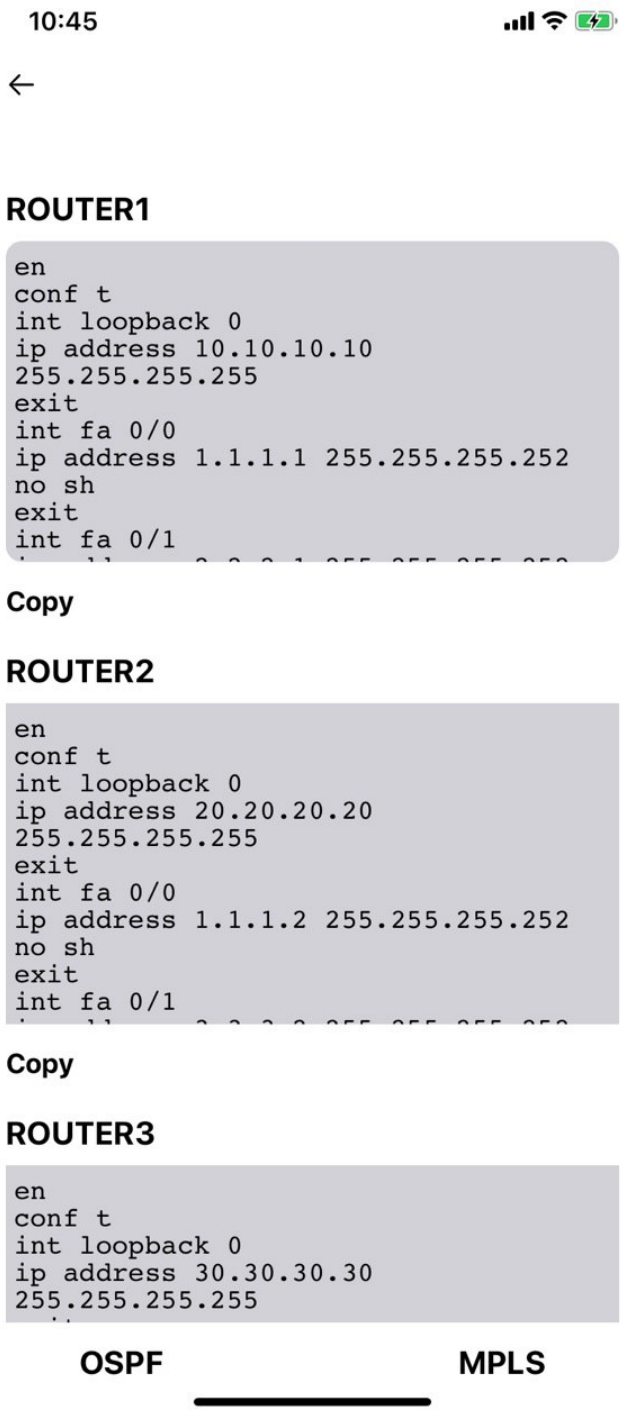

Рисунок 3.3 – Екран з командами для налаштування

На рис. 3.3 бачимо текстові вікна з командами, кнопки «OSPF» та «MPLS»

3.2 Опис функціоналу додатка

Користувач має ввести ip-адреси та маски для кожного роутера.

Рисунок 3.4 – Введення данних

При натисканні на необхідний роутер маємо можливість ввести дані для нього

Рисунок 3.5 – Обрано роутер R2

Можна скористатися кнопкою «Default settings», після натискання на неї всі поля заповняться даними. Після заповнення усіх даних користувач може натиснути кнопку «Next». Після натискання на неї користувач потрапляє на наступний екран. Див. рисунок 3.6

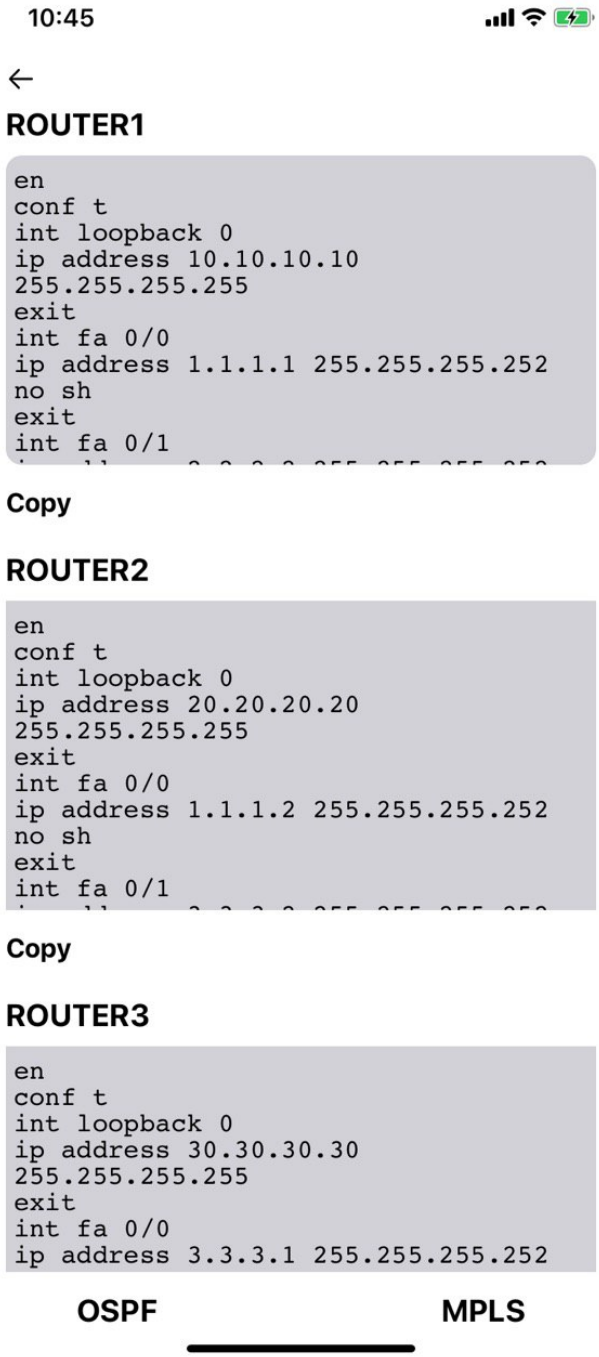

Рисунок 3.6 – Екран з командами налаштування OSPF

На рис. 3.6 можемо бачити текстові вікна з налаштуваннями для кожного роутера, їх можна проскролити щоб побачити всі необхідні

команди, а також скористатися кнопкою «copy» щоб скопіювати всі команди до буфера обміну. Знизу кнопки які перемикають налаштування OSPF та MPLS.

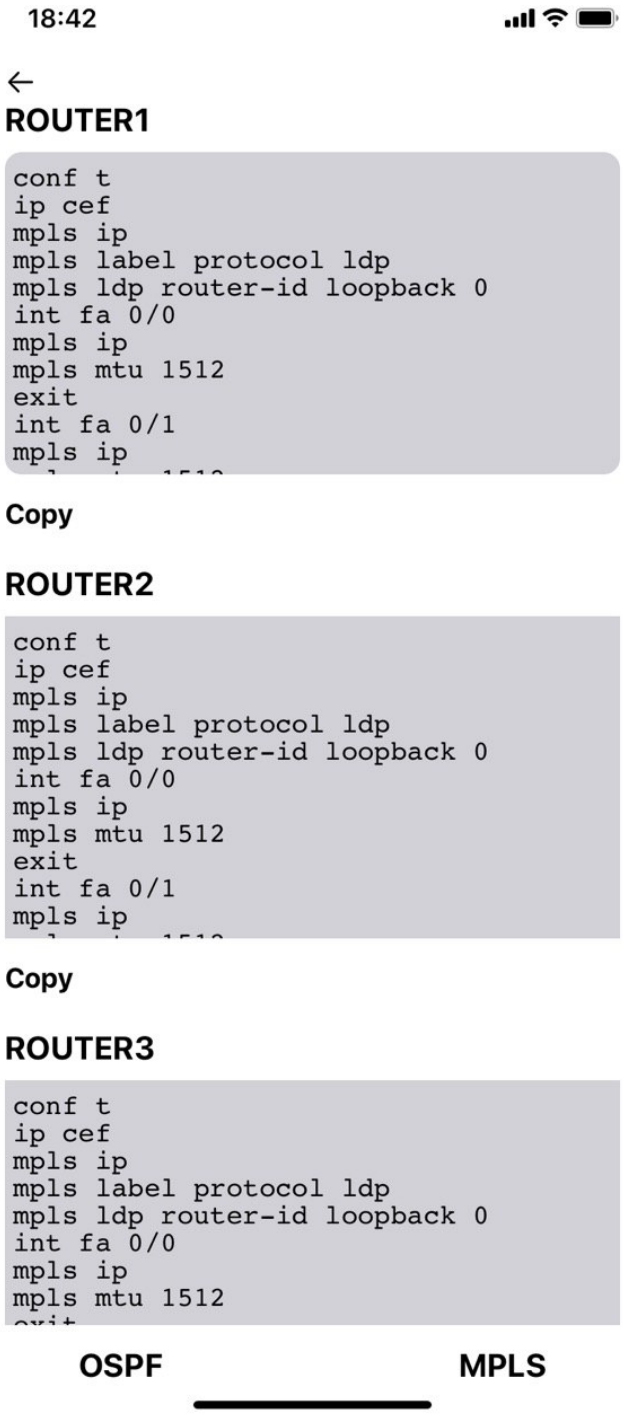

Рисунок 3.7 Екран з командами налаштування MPLS

При натисканні на кнопку «copy» бачимо повідомлення що дані скопійовані див. рисунок 3.8

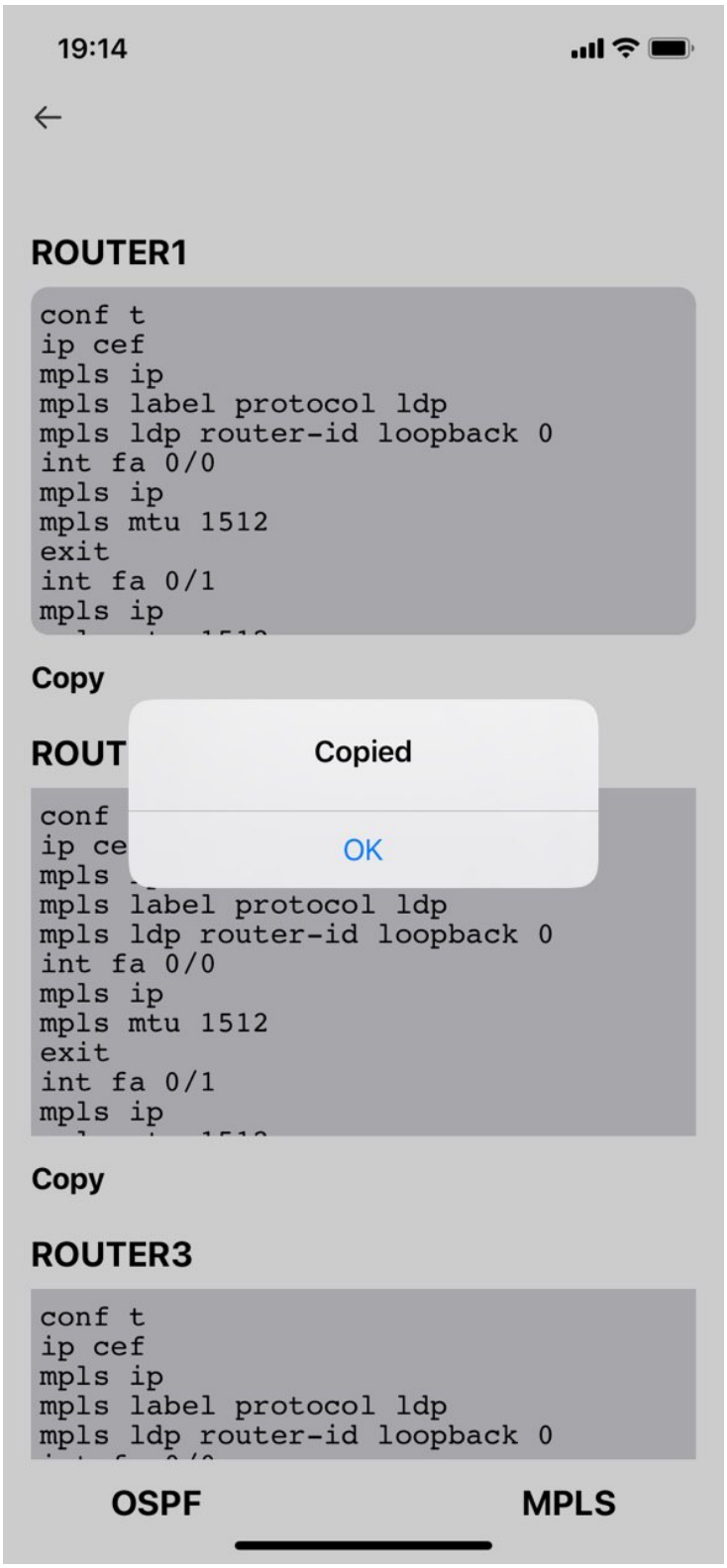

Рисунок 3.8 – Екран з повідомленням

Далі можна налаштувати спільний буфер обміну між телефоном та комп'ютером для швидкого копіювання одразу на роутери.

3.3 Тестування додатку

Відкриємо додаток та заповнимо усі необхідні поля, натиснувши «Default settings» див. рисунок 3.9

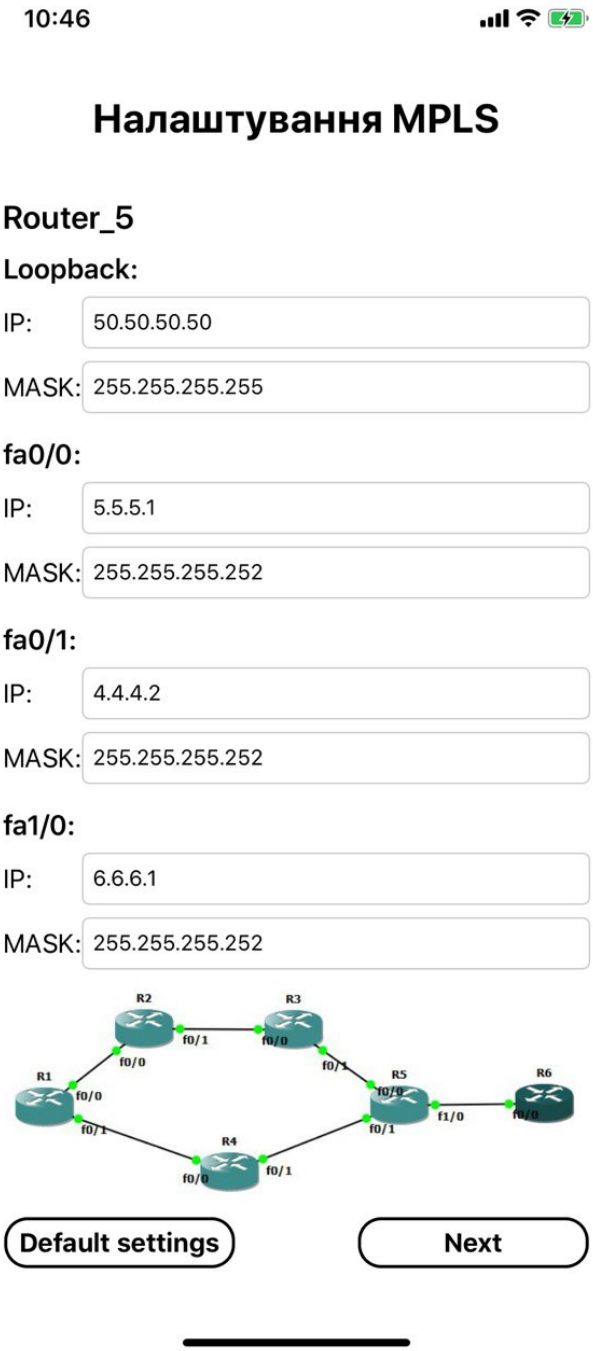

Рисунок 3.9 – Заповнення полів

Після цього натискаємо кнопку «Next». Та переходимо на наступний екран з командами для налаштування роутерів. Рис. 3.10

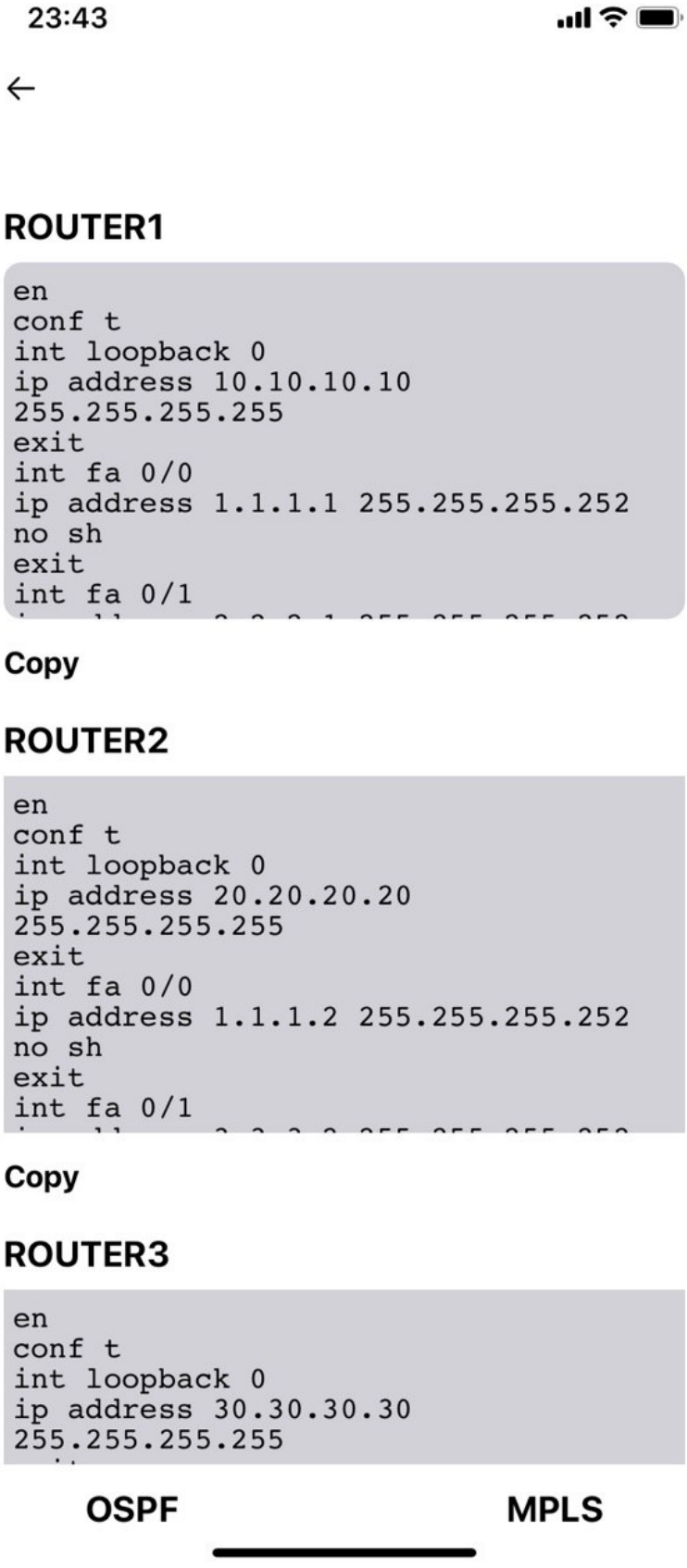

Рисунок 3.10 – Команди налаштування

Далі копіюємо команди і налаштовуємо схему в GNS3. Спочатку налаштуємо OSPF

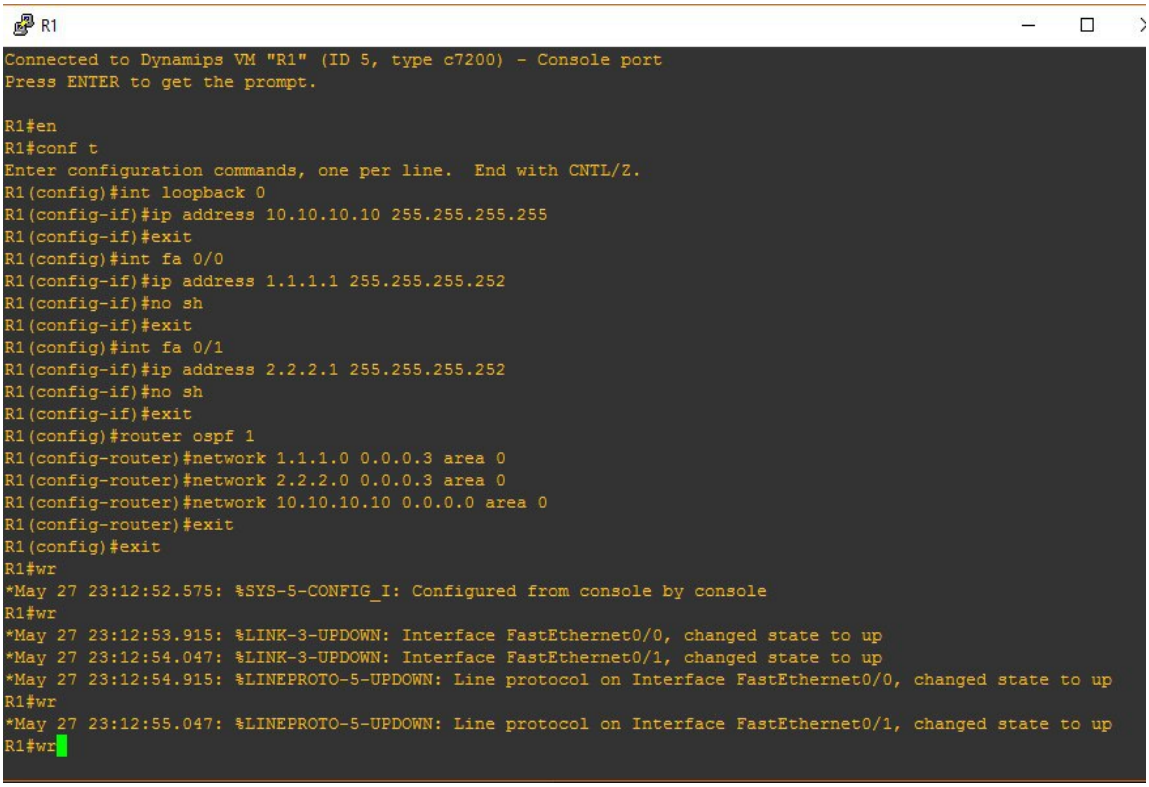

Рисунок 3.11 – Команди для налаштування OSPF

На рис. 3.11 ми скопіювали команди з додатку та вставили у консоль роутера R1, виконаємо ці дії для кожного роутера. Далі налаштовуємо MPLS

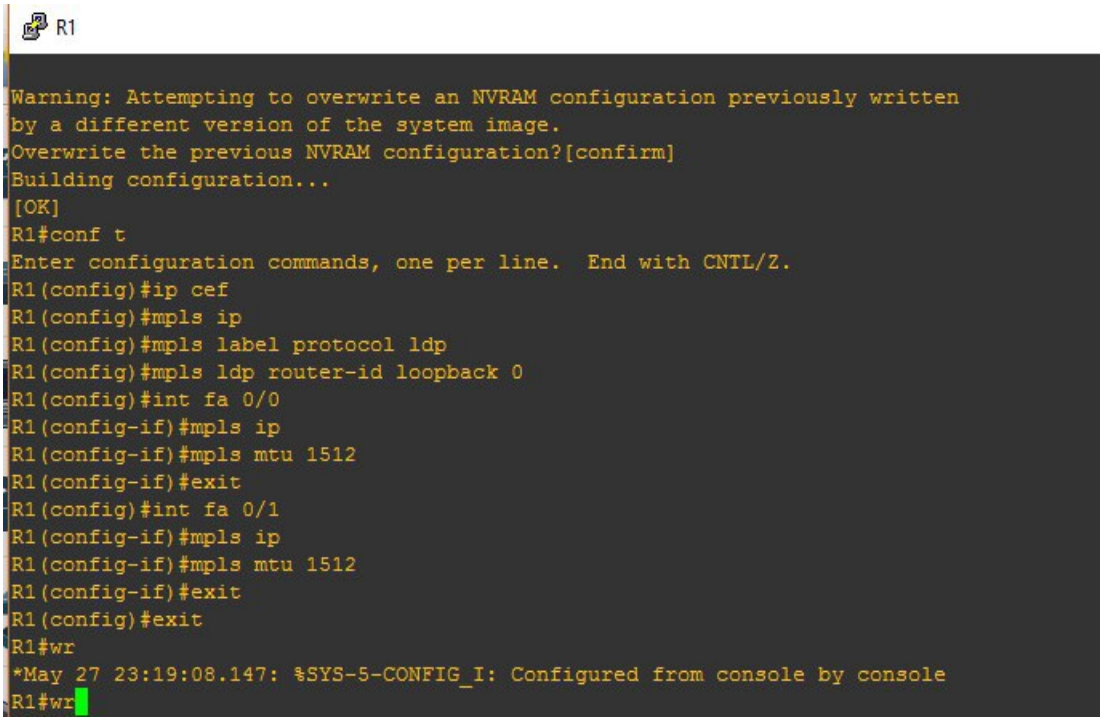

Рисунок 3.12 – Команди для налаштування MPLS

Перевіряємо командою sh mpls forwarding-table

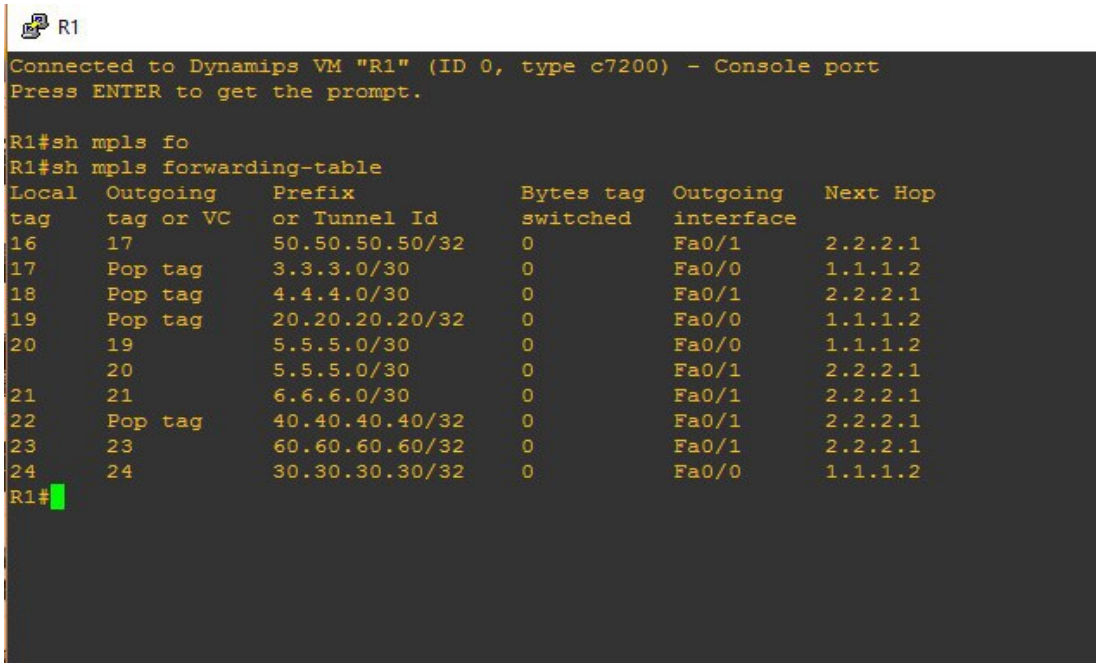

Рисунок 3.13 – Результат команди sh mpls forwarding-table

Local tag – локальна мітка, Outgoing tag or VC – яку мітку передає роутер наступному. Перший роутер з міткою 16 передає на другий роутер мітку 17. На роутері - 2 (другий рядок) локальна мітка 17, це та яку передав першій роутер, а вихідна мітка «Pop tag» це означає що роутер повинен зняти мітку.

Проаналізувавши дану таблицю бачимо що MPLS налаштована вірно.

ВИСНОВКИ

У ході виконання роботи був з’ясований принцип та необхідніть роботи MPLS. MPLS знижує навантаження на CPU маршрутизаторів, а також необхідний для налаштування VPN.

MPLS мережа є масштабованою, тобто можна відносно легко змінювати її дизайн, та протокол-незалежною, тобто вона може використовуватися, як транспорт для різноманітних IT протоколів. У MPLS мережі пакетам даних присвоюються спеціальні мітки.

Багатопротокольні технології MPLS полягає в тому, що вона дозволяє використовувати протоколи маршрутизації не тільки стека TCP / IP, але і будь-якого іншого стека, наприклад IPX / SPX.

Була побудована схема мережі з MPLS у емуляторі GNS3. Проаналізувавши схему та її принципові налаштування був розроблений мобільний додаток який дозволяє легко налаштувати мережу з використанням протоколу динамічної маршрутизації OSPF, та налаштування MPLS поверх неї. У ході роботи додаток був протестований, а саме, була налаштована схема з використанням додатку.

Додаток зручний та їм не складно користуватись. Використання додатку значно скорочує час побудови схеми.

СПИСОК ЛІТЕРАТУРИ

1. Протоколи і стандарти. MPLS як працює на навіщо потрібен [Електронний ресурс] - https://wiki.merionet.ru/seti/25/mpls-kak-rabotaet-i-zachem-nuzhen/
2. Cisco Packet Tracer – багатофункціональна програма моделювання мереж. [Електронний ресурс] - https://www.cisco.com/c/ru_ua/training-events/netacad/training-courses/cisco-packet-tracer.html
3. GNS3. Graphical Network Simulator [Електронний ресурс] - https://habr.com/ru/post/266503/
4. EIGRP. Протокол динамічної маршрутизації [Електронний ресурс] - https://ru.wikipedia.org/wiki/EIGRP
5. OSPF. Протокол динамічної маршрутизації [Електронний ресурс] - https://wiki.merionet.ru/seti/3/ospf/
6. Whireshark – програма аналізатор трафіка мереж Ethernet [Електронний ресурс] - https://ru.wikipedia.org/wiki/Wireshark
7. Objective-C – мова програмування [Електронний ресурс] - https://uk.wikipedia.org/wiki/Objective-C
8. Swift – мова програмування [Електронний ресурс] - https://uk.wikipedia.org/wiki/Swift_(мова_програмування)#cite_note-tnw-1
9. MPLS зробить маршрутизатор швидким [Електронний ресурс] - https://compress.ru/article.aspx?id=10621
10. В. Олифер, Н. Олифер "Компьютерные сети. Принципы, технологии, протоколы. Учебник" 2016. – 233 с.
11. Э. Таненбаум, Д. Уэзеролл "Компьютерные сети" 5-е изд. 2016. – 95 с.

ДОДАТКИ

Додаток А

```
import UIKit

enum Router: Int {
    case first, second, third, fourth
}

class SetupRoutersViewController: UIViewController {

    @IBOutlet weak var loopBackIPTextField: UITextField!
    @IBOutlet weak var loopBackMaskTextField: UITextField!

    @IBOutlet weak var fa00IpTextField: UITextField!
    @IBOutlet weak var fa00MaskTextField: UITextField!

    @IBOutlet weak var fa01IpTextField: UITextField!
    @IBOutlet weak var fa01MaskTextField: UITextField!

    @IBOutlet weak var titleRouterLabel: UILabel!
    @IBOutlet weak var defaultSettingsButton: UIButton!
    @IBOutlet weak var nextButton: UIButton!

    var currentRouter = Router.first
    var routerModels = RouterModel.emptyModels

    override func viewDidLoad() {
        super.viewDidLoad()
        hideKeyboard()
        setupButtons()

        loopBackIPTextField.delegate = self
        loopBackMaskTextField.delegate = self

        fa00IpTextField.delegate = self
        fa00MaskTextField.delegate = self

        fa01IpTextField.delegate = self
        fa01MaskTextField.delegate = self
    }

    @IBAction func didTapRouter1(_ sender: Any) {
        titleRouterLabel.text = "Router_1"

        currentRouter = .first
        let router = currentRouter
```

```
        let routerModel = routerModels.first { $0.router ==
router }!

        setupTextFieldsByRouterModel(routerModel: routerModel)

    }

    @IBAction func didTapRouter2(_ sender: Any) {
        titleRouterLabel.text = "Router_2"

        currentRouter = .second
        let router = currentRouter
        let routerModel = routerModels.first { $0.router ==
router }!

        setupTextFieldsByRouterModel(routerModel: routerModel)
    }

    @IBAction func didTapRouter3(_ sender: Any) {
        titleRouterLabel.text = "Router_3"

        currentRouter = .third
        let router = currentRouter
        let routerModel = routerModels.first { $0.router ==
router }!

        setupTextFieldsByRouterModel(routerModel: routerModel)
    }

    @IBAction func didTapRouter4(_ sender: Any) {
        titleRouterLabel.text = "Router_4"

        currentRouter = .fourth
        let router = currentRouter
        let routerModel = routerModels.first { $0.router ==
router }!

        setupTextFieldsByRouterModel(routerModel: routerModel)
    }

    @IBAction func didTapDefaultSettings(_ sender: Any) {
        routerModels = RouterModel.defaultModels

        let router = currentRouter
        let routerModel = routerModels.first { $0.router ==
router }!

        setupTextFieldsByRouterModel(routerModel: routerModel)
    }

    @IBAction func didTapNext(_ sender: Any) {
```

```
        let resultVC =
ResultViewController.storyboardViewController()
        resultVC.routerModels = self.routerModels

self.navigationController?.pushViewController(resultVC,
animated: true)
    }


    private func setupButtons() {
        nextButton.layer.borderColor = UIColor.black.cgColor
        nextButton.layer.borderWidth = 2
        nextButton.layer.cornerRadius = 15
        nextButton.titleEdgeInsets = UIEdgeInsets(top: 0,
left: 5, bottom: 0, right: 5)

        defaultSettingsButton.layer.borderColor =
UIColor.black.cgColor
        defaultSettingsButton.layer.borderWidth = 2
        defaultSettingsButton.layer.cornerRadius = 15
        defaultSettingsButton.titleEdgeInsets =
UIEdgeInsets(top: 0, left: 5, bottom: 0, right: 5)
    }

    private func setupTextFieldsByRouterModel(routerModel:
RouterModel) {
        loopBackIPTextField.text = routerModel.loopBackIP
        loopBackMaskTextField.text = routerModel.loopBackMask

        fa00IpTextField.text = routerModel.fa00Ip
        fa00MaskTextField.text = routerModel.fa00Mask

        fa01IpTextField.text = routerModel.fa01Ip
        fa01MaskTextField.text = routerModel.fa01Mask
    }
}

extension SetupRoutersViewController: UITextFieldDelegate {

    func textFieldShouldReturn(_ textField: UITextField) ->
Bool {
        textField.resignFirstResponder()
        return true
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        let router = currentRouter
        let routerModel = routerModels.first { $0.router ==
router }!

        switch textField {
```

```
        case loopBackIPTextField:
            routerModel.loopBackIP = textField.text ?? ""
        case loopBackMaskTextField:
            routerModel.loopBackMask = textField.text ?? ""
        case fa00IpTextField:
            routerModel.fa00Ip = textField.text ?? ""
        case fa00MaskTextField:
            routerModel.fa00Mask = textField.text ?? ""
        case fa01IpTextField:
            routerModel.fa01Ip = textField.text ?? ""
        case fa01MaskTextField:
            routerModel.fa01Mask = textField.text ?? ""
        default:
            break
        }
    }
     }
```

Додаток Б

```
import UIKit

class ResultViewController: UIViewController {

    @IBOutlet weak var firstRouterTextView: UITextView!
    @IBOutlet weak var secondRouterTextView: UITextView!
    @IBOutlet weak var thirdRouterTextView: UITextView!
    @IBOutlet weak var fourthRouterTextView: UITextView!

    var routerModels: [RouterModel]?

    override func viewDidLoad() {
        super.viewDidLoad()

self.navigationController?.interactivePopGestureRecognizer?.de
legate = self
        setupOSPFRouters()
    }

    @IBAction func didTapCopy(_ sender: UIButton) {
        switch sender.tag {
        case 0:
            copyComands(routerTextView: firstRouterTextView)
        case 1:
            copyComands(routerTextView: secondRouterTextView)
        case 2:
            copyComands(routerTextView: thirdRouterTextView)
        case 3:
            copyComands(routerTextView: fourthRouterTextView)
        default:
            break
```

```
        }
        self.showAlert(alertText: "Copied", alertMessage: "")
    }

    @IBAction func didTapOSPF(_ sender: Any) {
        setupOSPFRouters()
    }

    @IBAction func didTapMPLS(_ sender: Any) {
        setupMPLSComands(routerTextView: firstRouterTextView)
        setupMPLSComands(routerTextView: secondRouterTextView)
        setupMPLSComands(routerTextView: thirdRouterTextView)
        setupMPLSComands(routerTextView: fourthRouterTextView)
    }

    @IBAction func didTapBack(_ sender: Any) {
        self.navigationController?.popViewController(animated:
true)
    }

    private func setupOSPFRouters() {
        guard let routerModels = routerModels else { return }

        let firstRouterModel = routerModels.first { $0.router
== .first }!
        setupOSPFComands(routerModel: firstRouterModel,
routerTextView: firstRouterTextView)

        let secondRouterModel = routerModels.first { $0.router
== .second }!
        setupOSPFComands(routerModel: secondRouterModel,
routerTextView: secondRouterTextView)

        let thirdRouterModel = routerModels.first { $0.router
== .third}!
        setupOSPFComands(routerModel: thirdRouterModel,
routerTextView: thirdRouterTextView)

        let fourhtRouterModel = routerModels.first { $0.router
== .fourth}!
        setupOSPFComands(routerModel: fourhtRouterModel,
routerTextView: fourthRouterTextView)
    }

    private func setupOSPFComands(routerModel: RouterModel,
routerTextView: UITextView) {
        var text = ""
        text.append("en\n")
        text.append("conf t\n")
        text.append("int loopback 0\n")
```

```
        text.append("ip address \(routerModel.loopBackIP)
\(routerModel.loopBackMask)\n")
        text.append("exit\n")
        text.append("int fa 0/0\n")
        text.append("ip address \(routerModel.fa00Ip)
\(routerModel.fa00Mask)\n")
        text.append("no sh\n")
        text.append("exit\n")
        text.append("int fa 0/1\n")
        text.append("ip address \(routerModel.fa01Ip)
\(routerModel.fa01Mask)\n")
        text.append("no sh\n")
        text.append("exit\n")
        text.append("router ospf 1\n")
        text.append("network \(getNetwork(ip:
routerModel.fa00Ip)) \(backwardMask(mask:
routerModel.fa00Mask)) area 0\n")
        text.append("network \(getNetwork(ip:
routerModel.fa01Ip)) \(backwardMask(mask:
routerModel.fa01Mask)) area 0\n")
        text.append("network \(routerModel.loopBackIP)
\(backwardMask(mask: routerModel.loopBackMask)) area 0\n")
        text.append("exit\n")
        text.append("exit\n")
        text.append("wr\n")
        routerTextView.text = text
    }

    private func setupMPLSComands(routerTextView: UITextView)
{
        var text = ""
        text.append("conf t\n")
        text.append("ip cef\n")
        text.append("mpls ip\n")
        text.append("mpls label protocol ldp\n")
        text.append("mpls ldp router-id loopback 0\n")
        text.append("int fa 0/0\n")
        text.append("mpls ip\n")
        text.append("mpls mtu 1512\n")
        text.append("exit\n")
        text.append("int fa 0/1\n")
        text.append("mpls ip\n")
        text.append("mpls mtu 1512\n")
        text.append("exit\n")
        text.append("exit\n")
        text.append("wr\n")
        routerTextView.text = text
    }

    private func getNetwork(ip: String) -> String {
```

```
        var ipBlocks = ip.components(separatedBy:
[".").compactMap(Int.init).dropLast()

        ipBlocks.append(0)

        let networkString =
ipBlocks.compactMap(String.init).joined(separator: ".")

        return networkString
    }

    private func backwardMask(mask: String) -> String {
        let maskBlocks = mask.components(separatedBy:
[".").compactMap(Int.init)

        let backwardMaskBlocks = maskBlocks.map { -($0 - 255)
}

        let backwardMaskString =
backwardMaskBlocks.compactMap(String.init).joined(separator:
".")

        return backwardMaskString
    }

    private func copyComands(routerTextView: UITextView) {
        UIPasteboard.general.string = routerTextView.text
    }
}

extension ResultViewController: Storyboardable {

    static var storyboardName: String {
        "Main"
    }

}

extension ResultViewController: UIGestureRecognizerDelegate {

    func gestureRecognizer(_ gestureRecognizer:
UIGestureRecognizer, shouldBeRequiredToFailBy
otherGestureRecognizer: UIGestureRecognizer) -> Bool {
        return true
    }
     }
```

Додаток В

```
import Foundation
```

```
class RouterModel {

    let router: Router

    var loopBackIP: String
    var loopBackMask: String

    var fa00Ip: String
    var fa00Mask: String

    var fa01Ip: String
    var fa01Mask: String

    init(router: Router, loopBackIP: String, loopBackMask:
String, fa00Ip: String, fa00Mask: String, fa01Ip: String,
fa01Mask: String) {
        self.router = router

        self.loopBackIP = loopBackIP
        self.loopBackMask = loopBackMask

        self.fa00Ip = fa00Ip
        self.fa00Mask = fa00Mask

        self.fa01Ip = fa01Ip
        self.fa01Mask = fa01Mask
    }

    static var defaultModels: [RouterModel] {
        var defaultModels = [RouterModel]()
        defaultModels.append(RouterModel(router: .first,
loopBackIP: "10.10.10.10", loopBackMask: "255.255.255.255",
fa00Ip: "1.1.1.1", fa00Mask: "255.255.255.252", fa01Ip:
"2.2.2.1", fa01Mask: "255.255.255.252"))
        defaultModels.append(RouterModel(router: .second,
loopBackIP: "20.20.20.20", loopBackMask: "255.255.255.255",
fa00Ip: "1.1.1.2", fa00Mask: "255.255.255.252", fa01Ip:
"3.3.3.2", fa01Mask: "255.255.255.252"))
        defaultModels.append(RouterModel(router: .third,
loopBackIP: "30.30.30.30", loopBackMask: "255.255.255.255",
fa00Ip: "2.2.2.2", fa00Mask: "255.255.255.252", fa01Ip:
"4.4.4.2", fa01Mask: "255.255.255.252"))
        defaultModels.append(RouterModel(router: .fourth,
loopBackIP: "40.40.40.40", loopBackMask: "255.255.255.255",
fa00Ip: "3.3.3.1", fa00Mask: "255.255.255.252", fa01Ip:
"4.4.4.1", fa01Mask: "255.255.255.252"))
        return defaultModels
    }

    static var emptyModels: [RouterModel] {
        var emptyModels = [RouterModel]()
```

```
        emptyModels.append(RouterModel(router: .first,
loopBackIP: "", loopBackMask: "", fa00Ip: "", fa00Mask: "",
fa01Ip: "", fa01Mask: ""))
        emptyModels.append(RouterModel(router: .second,
loopBackIP: "", loopBackMask: "", fa00Ip: "", fa00Mask: "",
fa01Ip: "", fa01Mask: ""))
        emptyModels.append(RouterModel(router: .third,
loopBackIP: "", loopBackMask: "", fa00Ip: "", fa00Mask: "",
fa01Ip: "", fa01Mask: ""))
        emptyModels.append(RouterModel(router: .fourth,
loopBackIP: "", loopBackMask: "", fa00Ip: "", fa00Mask: "",
fa01Ip: "", fa01Mask: ""))
        return emptyModels
    }
     }
```

Додаток Д

```
R1:
en
conf t
int loopback 0
ip address 10.10.10.10 255.255.255.255
exit
int fa 0/0
ip address 1.1.1.1 255.255.255.252
no sh
exit
int fa 0/1
ip address 2.2.2.2 255.255.255.252
no sh
exit
router ospf 1
network 1.1.1.0 0.0.0.3 area 0
network 2.2.2.0 0.0.0.3 area 0
network 10.10.10.10 0.0.0.0 area 0
exit
exit
wr

R2:
en
conf t
int loopback 0
```

```
ip address 20.20.20.20 255.255.255.255
exit
int fa 0/0
ip address 1.1.1.2 255.255.255.252
no sh
exit
int fa 0/1
ip address 3.3.3.2 255.255.255.252
no sh
exit
router ospf 1
network 1.1.1.0 0.0.0.3 area 0
network 3.3.3.0 0.0.0.3 area 0
network 20.20.20.20 0.0.0.0 area 0
exit
exit
wr

R3:
en
conf t
int loopback 0
ip address 30.30.30.30 255.255.255.255
exit
int fa 0/0
ip address 3.3.3.1 255.255.255.252
no sh
exit
int fa 0/1
ip address 5.5.5.2 255.255.255.252
no sh
exit
router ospf 1
network 3.3.3.0 0.0.0.3 area 0
network 5.5.5.0 0.0.0.3 area 0
network 30.30.30.30 0.0.0.0 area 0
exit
exit
wr
```

```
R4:
en
conf t
int loopback 0
ip address 40.40.40.40 255.255.255.255
exit
int fa 0/0
ip address 2.2.2.1 255.255.255.252
no sh
exit
int fa 0/1
ip address 4.4.4.1 255.255.255.252
no sh
exit
router ospf 1
network 2.2.2.0 0.0.0.3 area 0
network 4.4.4.0 0.0.0.3 area 0
network 40.40.40.40 0.0.0.0 area 0
exit
exit
wr

R5:
en
conf t
int loopback 0
ip address 50.50.50.50 255.255.255.255
exit
int fa 0/0
ip address 5.5.5.1 255.255.255.252
no sh
exit
int fa 0/1
ip address 4.4.4.2 255.255.255.252
no sh
exit
int fa 1/0
ip address 6.6.6.1 255.255.255.252
```

```
no sh
exit
router ospf 1
network 5.5.5.0 0.0.0.3 area 0
network 4.4.4.0 0.0.0.3 area 0
network 6.6.6.0 0.0.0.3 area 0
network 50.50.50.50 0.0.0.0 area 0
exit
exit
wr

R6:
en
conf t
int loopback 0
ip address 60.60.60.60 255.255.255.255
exit
int fa 0/0
ip address 6.6.6.2 255.255.255.252
no sh
exit
router ospf 1
network 6.6.6.0 0.0.0.3 area 0
network 60.60.60.60 0.0.0.0 area 0
exit
exit
wr
```